**Method Overriding Example**

Lets take a simple example to understand this. We have two classes: A child class Boy and a parent class Human. The Boy class extends Human class. Both the classes have a common method void eat(). Boy class is giving its own implementation to the eat() method or in other words it is overriding the eat() method.

The purpose of Method Overriding is clear here. Child class wants to give its own implementation so that when it calls this method, it prints Boy is eating instead of Human is eating.

class Human{

//Overridden method

public void eat()

{

System.out.println("Human is eating");

}

}

class Boy extends Human{

//Overriding method

public void eat(){

System.out.println("Boy is eating");

}

public static void main( String args[]) {

Boy obj = new Boy();

//This will call the child class version of eat()

obj.eat();

}

}

### Overriding example

package beginnersbook.com;

class CarClass

{

public int speedLimit()

{

return 100;

}

}

class Ford extends CarClass

{

public int speedLimit()

{

return 150;

}

public static void main(String args[])

{

CarClass obj = new Ford();

int num= obj.speedLimit();

System.out.println("Speed Limit is: "+num);

}

}

Output:

**Why we need an abstract class?**

Lets say we have a class Animal that has a method sound() and the subclasses(see [inheritance](https://beginnersbook.com/2013/03/inheritance-in-java/)) of it like Dog, Lion, Horse, Cat etc. Since the animal sound differs from one animal to another, there is no point to implement this method in parent class. This is because every child class must override this method to give its own implementation details, like Lion class will say “Roar” in this method and Dog class will say “Woof”.

So when we know that all the animal child classes will and should override this method, then there is no point to implement this method in parent class. Thus, making this method abstract would be the good choice as by making this method abstract we force all the sub classes to implement this method( otherwise you will get compilation error), also we need not to give any implementation to this method in parent class.

//abstract parent class

abstract class Animal{

//abstract method

public abstract void sound();

}

//Dog class extends Animal class

public class Dog extends Animal{

public void sound(){

System.out.println("Woof");

}

public static void main(String args[]){

Animal obj = new Dog();

obj.sound();

}

}

Output:

**Abstraction** is a process of hiding the implementation details and showing only functionality to the user.

Another way, it shows only essential things to the user and hides the internal details, for example, sending SMS where you type the text and send the message. You don't know the internal processing about the message delivery.

Abstraction lets you focus on what the object does instead of how it does it

1. **abstract** **class** Bike{
2. **abstract** **void** run();
3. }
4. **class** Honda4 **extends** Bike{
5. **void** run(){System.out.println("running safely");}
6. **public** **static** **void** main(String args[]){
7. Bike obj = **new** Honda4();
8. obj.run();
9. }
10. }
11. **abstract** **class** Shape{
12. **abstract** **void** draw();
13. }
14. //In real scenario, implementation is provided by others i.e. unknown by end user
15. **class** Rectangle **extends** Shape{
16. **void** draw(){System.out.println("drawing rectangle");}
17. }
18. **class** Circle1 **extends** Shape{
19. **void** draw(){System.out.println("drawing circle");}
20. }
21. //In real scenario, method is called by programmer or user
22. **class** TestAbstraction1{
23. **public** **static** **void** main(String args[]){
24. Shape s=**new** Circle1();//In a real scenario, object is provided through method, e.g., getShape() method
25. s.draw();
26. }
27. }

### Advantage of java inner classes

There are basically three advantages of inner classes in java. They are as follows:

1) Nested classes represent a special type of relationship that is **it can access all the members (data members and methods) of outer class** including private.

2) Nested classes are used **to develop more readable and maintainable code** because it logically group classes and interfaces in one place only.

3) **Code Optimization**: It requires less code to write.