Week 8 Java Friday March 5, 2021

**Method overriding in java**

Declaring a method in **sub class** which is already present in **parent class** is known as method overriding.

1. Overriding is done so that a child class can give its own implementation to a method which is already provided by the parent class.
2. In this case the method in parent class is called overridden method and the method in child class is called overriding method.
3. **Advantage of method overriding**

The main advantage of method overriding is that the class can give its own specific implementation to a inherited method **without even modifying the parent class code**.

## Method Overriding and Dynamic Method Dispatch

## Method Overriding is an example of [runtime polymorphism](https://beginnersbook.com/2013/04/runtime-compile-time-polymorphism/). When a parent class reference points to the child class object then the call to the overridden method is determined at runtime, because during method call which method(parent class or child class) is to be executed is determined by the type of object. This process in which call to the overridden method is resolved at runtime is known as dynamic method dispatch.

## Rules of method overriding in Java

1. Argument list: The argument list of overriding method (method of child class) must match the Overridden method(the method of parent class). The data types of the arguments and their sequence should exactly match.
2. [Access Modifier](https://beginnersbook.com/2013/05/java-access-modifiers/) of the overriding method (method of subclass) cannot be more restrictive than the overridden method of parent class.
3. private, static and final methods cannot be overridden as they are local to the class. However static methods can be re-declared in the sub class, in this case the sub-class method would act differently and will have nothing to do with the same static method of parent class.
4. Overriding method (method of child class) can throw [unchecked exceptions](https://beginnersbook.com/2013/04/java-checked-unchecked-exceptions-with-examples/), regardless of whether the overridden method(method of parent class) throws any exception or not. However the overriding method should not throw [checked exceptions](https://beginnersbook.com/2013/04/java-checked-unchecked-exceptions-with-examples/) that are new or broader than the ones declared by the overridden method. We will discuss this in detail with example in the upcoming tutorial.
5. Binding of overridden methods happen at runtime which is known as [dynamic binding](https://beginnersbook.com/2013/04/java-static-dynamic-binding/).
6. If a class is extending an [abstract class](https://beginnersbook.com/2013/05/java-abstract-class-method/) or implementing an [interface](https://beginnersbook.com/2013/05/java-interface/) then it has to override all the abstract methods unless the class itself is a abstract class.

## Super keyword in Method Overriding

The [super keyword](https://beginnersbook.com/2014/07/super-keyword-in-java-with-example/) is used for calling the parent class method/constructor.

 Super myMethod() calls the myMethod() method of base class while super() calls the [constructor](https://beginnersbook.com/2013/03/constructors-in-java/) of base class.

# Method Overloading in Java with examples

Method Overloading is a feature that allows a class to have more than one method having the same name, if their argument lists are different. It is similar to [constructor overloading](https://beginnersbook.com/2013/05/constructor-overloading/) in Java, that allows a class to have more than one constructor having different argument lists.

(For example the argument list of a method add(int a, int b) having two parameters is different from the argument list of the method add(int a, int b, int c) having three parameters.)

## Three ways to overload a method

## 1. Number of parameters.

**add(int, int)**

**add(int, int, int**

**2. Data type of parameters.**

**add(int, float)**

**add(float, int)**

**3.Sequence of Data type of parameters**

**add(int, float)**

**add(float, int)**

# Interface in java

Abstraction is a process where you show only “relevant” data and “hide” unnecessary details of an object from the user(See: [Abstraction](https://beginnersbook.com/2013/03/oops-in-java-encapsulation-inheritance-polymorphism-abstraction/)). In this guide, we will cover **what is an interface in java**, why we use it and what are rules that we must follow while using interfaces in [Java Programming](https://beginnersbook.com/java-tutorial-for-beginners-with-examples/).

Interface looks like a class but it is not a class. An interface can have methods and variables just like the class but the methods declared in interface are by default abstract (only method signature, no body, see: [Java abstract method](https://beginnersbook.com/2014/01/abstract-method-with-examples-in-java/))

# Polymorphism in Java

Polymorphism is one of the [OOPs](https://beginnersbook.com/2013/04/oops-concepts/) feature that allows us to perform a single action in different ways. For example, lets say we have a class Animal that has a method sound().

Polymorphism is the capability of a method to do different things based on the object that it is acting upon. In other words, polymorphism allows you define one interface and have multiple implementations

Types of polymorphism and method overloading & overriding are covered in the separate tutorials. You can refer them here:

1. [Method Overloading in Java](https://beginnersbook.com/2013/05/method-overloading/) – This is an example of compile time (or static polymorphism)  
2. [Method Overriding in Java](https://beginnersbook.com/2014/01/method-overriding-in-java-with-example/) – This is an example of runtime time (or dynamic polymorphism)  
3. [Types of Polymorphism – Runtime and compile time](https://beginnersbook.com/2013/04/runtime-compile-time-polymorphism/) – This is our next tutorial where we have covered the types of polymorphism in detail. I would recommend you to go though method overloading and overriding before going though this topic.

# Types of polymorphism in java-

# Runtime and Compile time polymorphism.

There are two types of polymorphism in java:  
1) **Static Polymorphism** also known as compile time polymorphism  
2) **Dynamic Polymorphism** also known as runtime polymorphism.

**Abstract Class in Java**

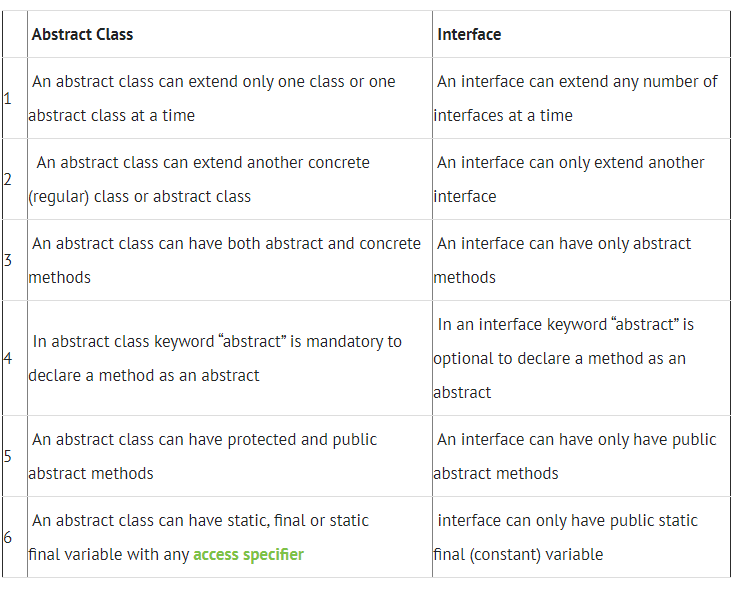
A class that is declared using “**abstract**” keyword is known as abstract class. It can have abstract methods(methods without body) as well as concrete methods (regular methods with body). A normal class(non-abstract class) cannot have abstract methods.

[Abstract method in Java](https://beginnersbook.com/2014/01/abstract-method-with-examples-in-java/)

For now lets just see some basics and example of abstract method.  
1) Abstract method has no body.  
2) Always end the declaration with a **semicolon**(;).  
3) It must be [overridden](https://beginnersbook.com/2014/01/method-overriding-in-java-with-example/). An abstract class must be extended and in a same way abstract method must be overridden.  
4) A class has to be declared abstract to have abstract methods.

## Rules of Abstract Method

1. Abstract methods don’t have body, they just have method signature as shown above.  
   2. If a class has an abstract method it should be declared abstract, the vice versa is not true, which means an abstract class doesn’t need to have an abstract method compulsory.  
   3. If a regular class extends an abstract class, then the class must have to implement all the abstract methods of abstract parent class or it has to be declared abstract as well.



## What is encapsulation?

encapsulation is to hide the implementation details from users. If a data member is private it means it can only be accessed within the same class. No outside class can access private data member (variable) of other class.

How to implement encapsulation in java:

1. Make the instance variables private so that they cannot be accessed directly from outside the class. You can only set and get values of these variables through the methods of the class.  
   2) Have getter and setter methods in the class to set and get the values of the fields.

**Advantages of encapsulation**

1. It improves maintainability and flexibility and re-usability: for e.g. In the above code the implementation code of void setEmpName(String name) and String getEmpName() can be changed at any point of time. Since the implementation is purely hidden for outside classes they would still be accessing the private field empName using the same methods (setEmpName(String name) and getEmpName()). Hence the code can be maintained at any point of time without breaking the classes that uses the code. This improves the re-usability of the underlying class.
2. The fields can be made read-only (If we don’t define setter methods in the class) or write-only (If we don’t define the getter methods in the class). For e.g. If we have a field(or variable) that we don’t want to be changed so we simply define the variable as private and instead of set and get both we just need to define the get method for that variable. Since the set method is not present there is no way an outside class can modify the value of that field.
3. User would not be knowing what is going on behind the scene. They would only be knowing that to update a field call set method and to read a field call get method but what these set and get methods are doing is purely hidden from them.