Miscellaneous

Method Signature

It consists of the method name and the parameter list (type and number of parameters). The return type and access modifiers (like public, private) are not part of the method signature.

Polymorphsim

Polymorphism is the ability of an object to take on many forms. It allows a common interface to be used for multiple forms (data types), enabling a method to behave differently based on the object that invokes it.

Types:

1. Compile-time Polymorphism (Static Polymorphism)

Polymorphism resolved(by compiler)during compilation. It happens when multiple methods have the same name but different parameters (method overloading).

Rules:

1. Method name must be same.
2. Parameter list must be different(Number, type or order)

Do Remember:

1. You cannot overload a method only by changing its return type.
2. Overloaded methods can have different access levels.
3. Method overloading can occur within the same class, or between a superclass and subclass.
4. Runtime Polymorphism (Dynamic Polymorphism)

Polymorphism resolved(by JVM) during execution. It happens when a subclass overrides a method of its superclass and Java uses the actual object type at runtime to decide which method to call.

Rules:

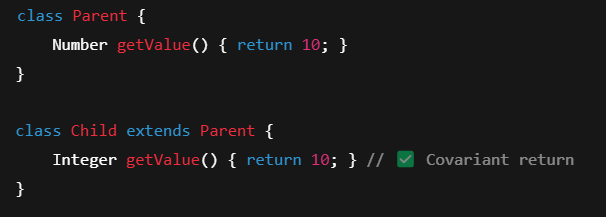
1. The method name, return type, and parameter list in the child class must exactly match the method in the parent class.
2. You can only override methods that are inherited from a superclass.

You cannot override:

1. Constructors
2. Static methods
3. Final methods
4. Private methods (they are not inherited)
5. The overridden method cannot have a more restrictive access modifier than the method in the parent class.

Ex. Parent class method is protected.

1. Since Java 5, it's legal to override a method and change the return type as long as it is a subtype of the original return type.



1. Overriding vs Hiding (for Static Methods)

Static methods are not overridden, they are hidden.

The method that gets called depends on the reference type, not the object type.

A screen shot of a computer program

AI-generated content may be incorrect.

Why Compile-Time Polymorphism is Called Static Polymorphism

Compile-time polymorphism is also known as static polymorphism because the decision about which method to call is made at compile time, before the program is actually run. This decision is "static" because it does not depend on the runtime type of objects, but rather on the method signatures and the context in which the methods are called.

Interface

1. All variables in an interface are implicitly public, static, and final.

Sout(InterfaceName.variableName); 🡪 No need to implement to access the variables.

These are not Instance variables.

1. Interfaces cannot contain instance fields or constructors.
2. Default Methods in Interfaces (Java 8 and later)

default methods allow you to define methods with default behavior in interfaces. This is useful for adding new methods to interfaces without breaking existing implementations.

default methods can have a body and can be overridden in the implementing class if desired.

1. All methods in an interface are implicitly abstract unless they are declared default or static
2. Interfaces vs Abstract Classes

A screenshot of a computer

AI-generated content may be incorrect.

Abstract Keyword

Abstract classes can hold **instance**, **static**, and **final variables**, just like concrete classes.

Abstract Classes: Classes that cannot be instantiated, but serve as a blueprint for subclasses.

Abstract Methods: Methods that do not have a body and must be implemented by subclasses.

Abstract methods in abstract classes must be implemented by its subclasses unless they are also abstract.

Can have constructors: Although abstract classes cannot be instantiated, they can have constructors that are called when a subclass is instantiated.