Build COMPETENCY
across your TEAM

SYNERGETICS GET IT RIGHT

Microsoft Partner
Gold Cloud Platform
Silver Learning

# Interfaces and Abstract Classes

Google Cloud Platform
Windows Azure

amazon web services · APACHE kafka A distributed streaming platform · Apache SOFTWARE FOUNDATION · APACHE Spark · Java · hadoop

## Context And Dependency Injection

Abstract class

Interfaces

default methods

static methods on Interface

Runtime Polymorphism

## Abstract class

## Abstract class

- A class which contains the **abstract** keyword in its declaration is known as abstract class.

## Some Rules

- Abstract classes may or may not contain *abstract methods*, i.e., methods without body ( public void get(); )
- But, if a class has at least one abstract method, then the class **must** be declared abstract.
- If a class is declared abstract, it cannot be instantiated.
- To use an abstract class, you have to inherit it from another class, provide implementations to the abstract methods in it.
- If you inherit an abstract class, you have to provide implementations to all the abstract methods in it.

## Inheriting the Abstract Class

- We can inherit the properties of Abstract class just like concrete class
- Abstract Classes are ideally created to be abstracted.

## Abstract Methods

- If you want a class to contain a particular method but you want the actual implementation of that method to be determined by child classes, you can declare the method in the parent class as an abstract.
  - **abstract** keyword is used to declare the method as abstract.
  - You have to place the **abstract** keyword before the method name in the method declaration.
  - An abstract method contains a method signature, but no method body.
  - Instead of curly braces, an abstract method will have a semoi colon (;) at the end.

## Consequences

- Declaring a method as abstract has two consequences –
  - The class containing it must be declared as abstract.
  - Any class inheriting the current class must either override the abstract method or declare itself as abstract.

SYNERGETICS GET IT RIGHT

Microsoft Partner
Gold Cloud Platform
Silver Learning

## Interfaces

- An interface is a reference type in Java. It is similar to class. It is a collection of abstract methods. A class implements an interface, thereby inheriting the abstract methods of the interface.
- Along with abstract methods, an interface may also contain constants, default methods, static methods, and nested types. Method bodies exist only for default methods and static methods.
- Writing an interface is similar to writing a class. But a class describes the attributes and behaviors of an object. And an interface contains behaviors that a class implements.
- Unless the class that implements the interface is abstract, all the methods of the interface need to be defined in the class.

## An interface is similar to a class in the following ways

- An interface can contain any number of methods.
- An interface is written in a file with a **.java** extension, with the name of the interface matching the name of the file.
- The byte code of an interface appears in a **.class** file.
- Interfaces appear in packages, and their corresponding bytecode file must be in a directory structure that matches the package name.

## An interface is different from a class in several ways, including

- You cannot instantiate an interface.
- An interface does not contain any constructors.
- All of the methods in an interface are abstract.
- An interface cannot contain instance fields. The only fields that can appear in an interface must be declared both static and final.
- An interface is not extended by a class; it is implemented by a class.
- An interface can extend multiple interfaces.

## Declaring Interfaces

- The **interface** keyword is used to declare an interface.

```
/* File name : NameOfInterface.java */
import java.lang.*;
// Any number of import statements

public interface NameOfInterface {
   // Any number of final, static fields
   // Any number of abstract method declarations\
}
```

## Interfaces have the following properties

- An interface is implicitly abstract. You do not need to use the **abstract**keyword while declaring an interface.
- Each method in an interface is also implicitly abstract, so the abstract keyword is not needed.
- Methods in an interface are implicitly public.

- One class can implement one of more interface
- One interface can extend any number of interfaces at a given time
- Interface cannot implement another interface
- Classes cannot extend an interface
- Class “implements” an interface

SYNERGETICS GET IT RIGHT

Microsoft Partner
Gold Cloud Platform
Silver Learning

## default methods

## Default Method

```
public interface oldInterface {
    public void existingMethod();
        default public void newDefaultMethod() {
        System.out.println("New default method is added in interface");
    }
}
```

## Interface Default Methods in Java 8

- Java 8 introduces “Default Method” or (Defender methods) new feature, which allows developer to add new methods to the interfaces without breaking the existing implementation of these interfaces.
- It provides flexibility to allow interface define implementation which will use as default in the situation where a concrete class fails to provide an implementation for that method.

## Why Default Method?

- Reengineering an existing JDK framework is always very complex.
- Modify one interface in JDK framework breaks all classes that extends the interface which means that adding any new method could break millions of lines of code.
- Therefore, default methods have introduced as a mechanism to extending interfaces in a backward compatible way.

## Why Default Method?

- Default methods can be provided to an interface without affecting implementing classes as it includes an implementation.
- If each added method in an interface defined with implementation then no implementing class is affected.
- An implementing class can override the default implementation provided by the interface.

- For Java 8, the JDK collections have been extended and forEach method is added to the entire collection (which work in conjunction with lambdas). With the conventional way, the code looks like below,

```
public interface Iterable<T> {
    public void forEach(Consumer<? super T> consumer);
}
```

## Some important points to discover

- When to Use Default Method Over Abstract Classes
- Abstract classes versus interfaces in Java 8
- Default Method and Multiple Inheritance Ambiguity Problems
- Difference Between Default Method and Regular Method

SYNERGETICS GET IT RIGHT

Microsoft Partner
Gold Cloud Platform
Silver Learning

## static methods on Interface

## Java Interface Static Method

- Java interface static method is similar to default method except that we can’t override them in the implementation classes.
- This feature helps us in avoiding undesired results incase of poor implementation in implementation classes.

## Important points about java interface static method:

- Java interface static method is part of interface, we can’t use it for implementation class objects.
- Java interface static methods are good for providing utility methods, for example null check, collection sorting etc.
- Java interface static method helps us in providing security by not allowing implementation classes to override them.

## Important points about java interface static method

- We can’t define interface static method for Object class methods, we will get compiler error as “This static method cannot hide the instance method from Object”. This is because it’s not allowed in java, since Object is the base class for all the classes and we can’t have one class level static method and another instance method with same signature.
- We can use java interface static methods to remove utility classes such as Collections and move all of it’s static methods to the corresponding interface, that would be easy to find and use.

SYNERGETICS GET IT RIGHT

Microsoft Partner
Gold Cloud Platform
Silver Learning

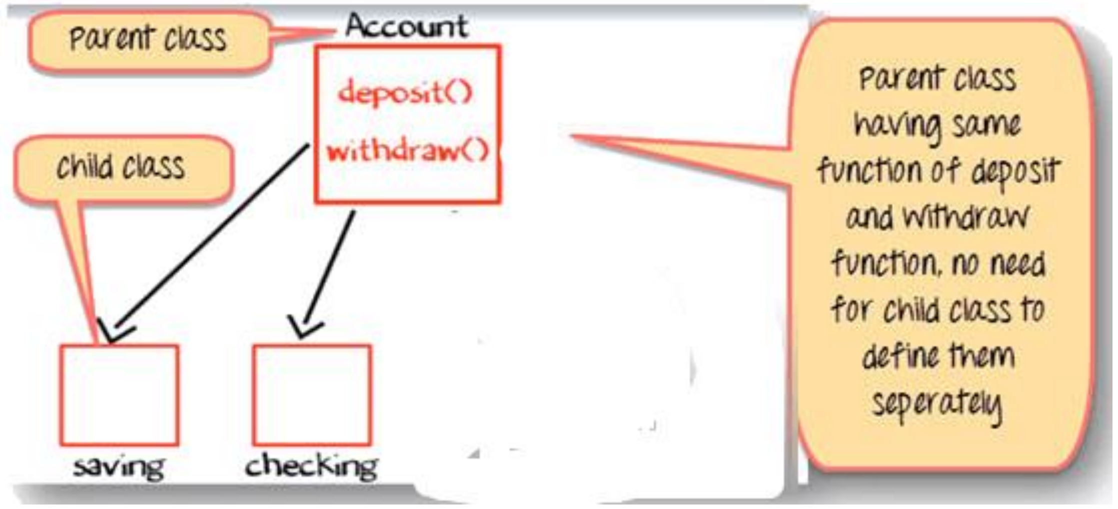
## Runtime Polymorphism

## What is Polymorphism?

- Polymorphism is a OOPs concept where one name can have many forms.
- For example, you have a smartphone for communication. The communication mode you choose could be anything. It can be a call, a text message, a picture message, mail, etc. So, the goal is common that is communication, but their approach is different. This is called **Polymorphism.**

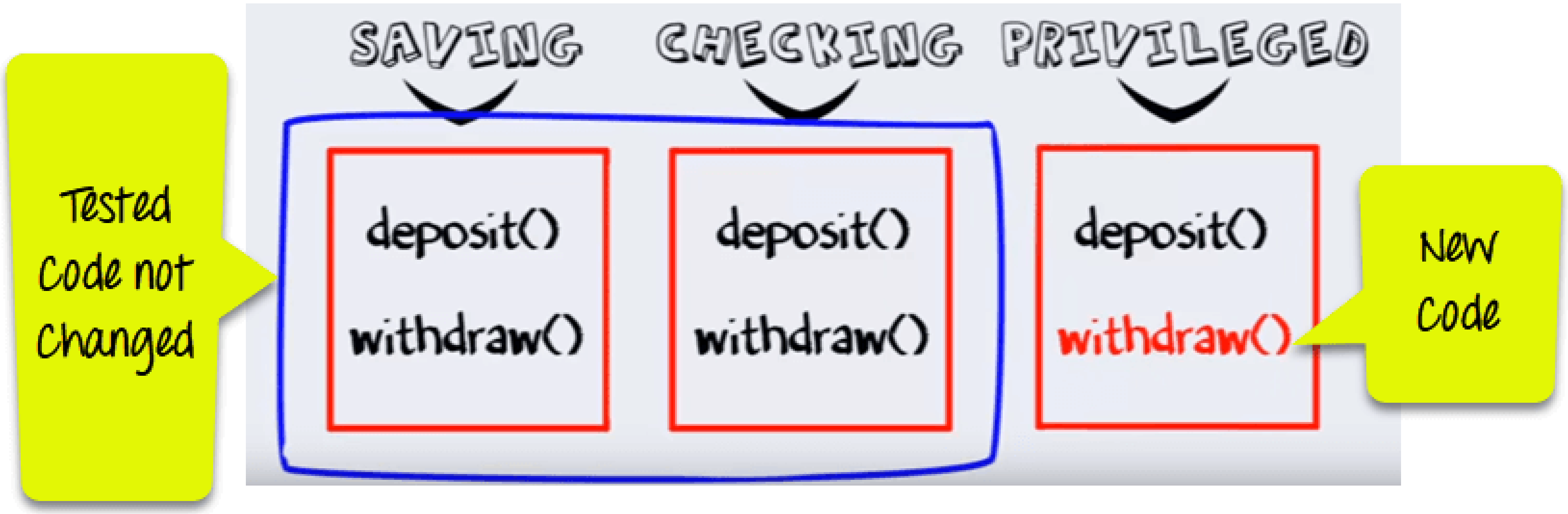
## Java Polymorphism in OOP's with Example

- We have one parent class, 'Account' with function of deposit and withdraw. Account has 2 child classes
- The operation of saving and withdraw is same for Saving and Checking accounts. So the inherited methods from Account class will work.

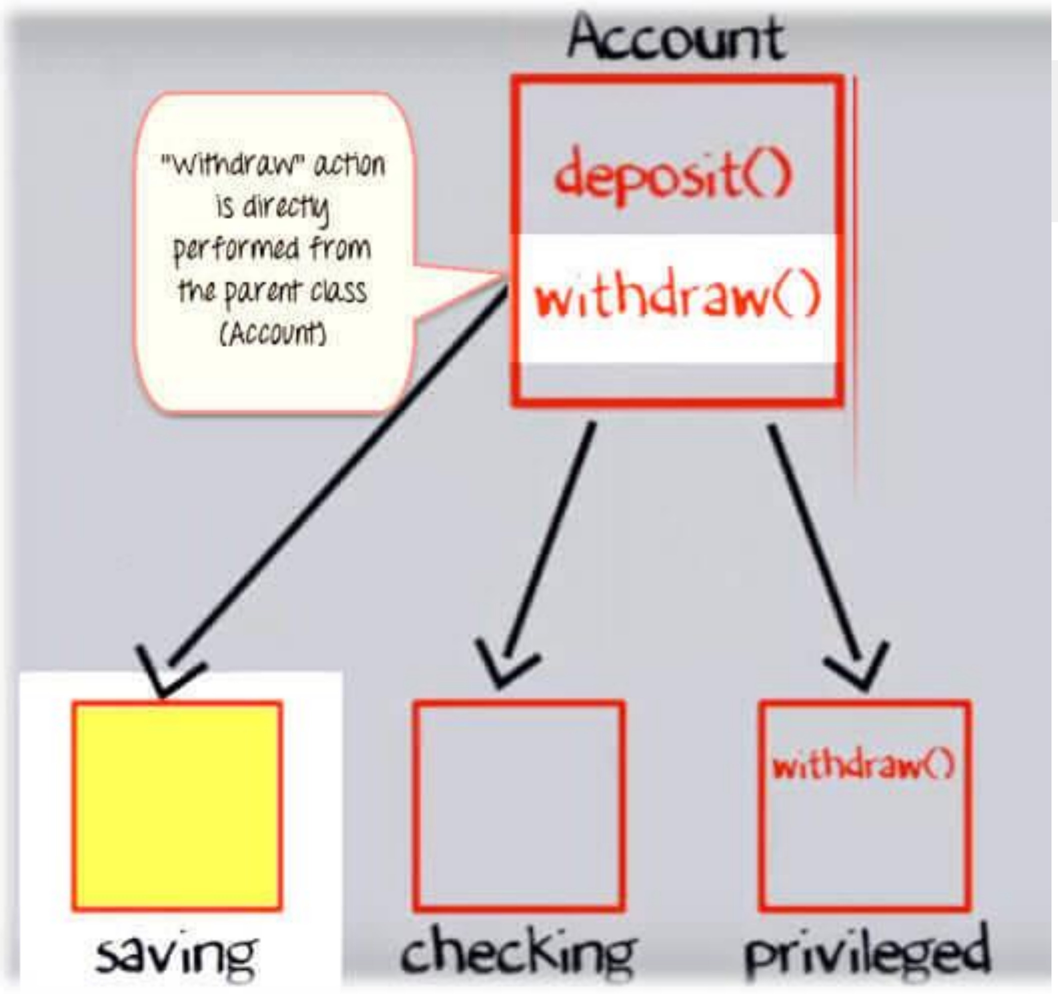

Parent class
Account
deposit()
withdraw()
Child class
saving
checking
Parent class having same function of deposit and withdraw function, no need for child class to define them seperately

## Change in Software Requirement

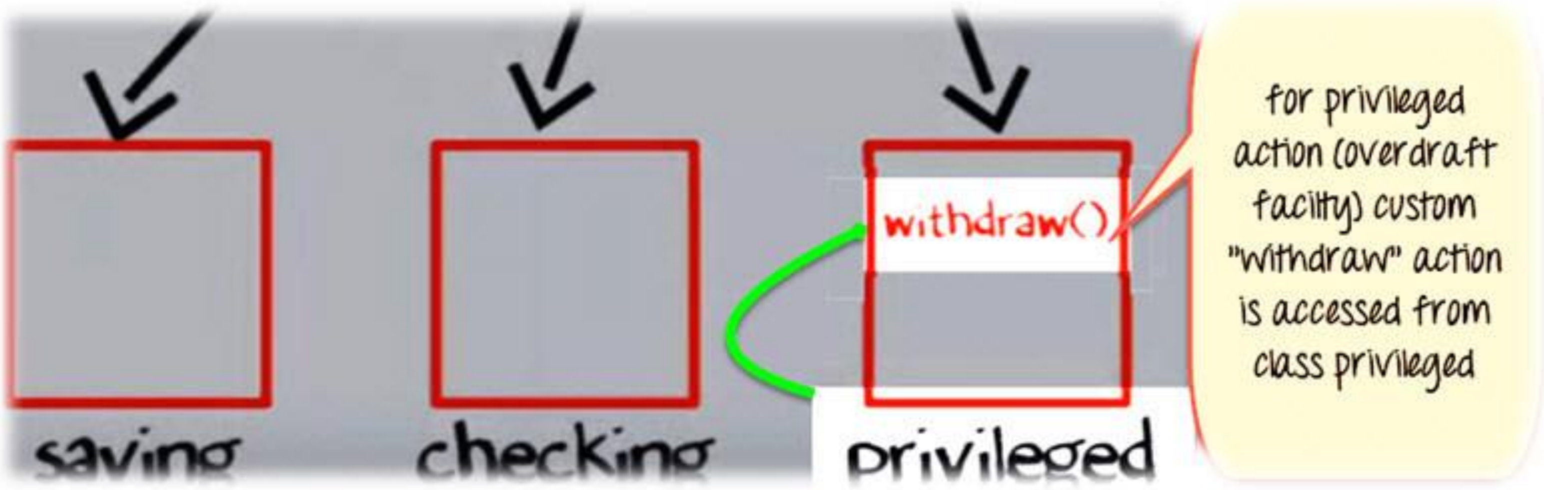
- There is a change in the requirement specification, something that is so common in the software industry. You are supposed to add functionality privileged Banking Account with Overdraft Facility.
- For a background, overdraft is a facility where you can withdraw an amount more than available the balance in your account.
- So, withdraw method for privileged needs to implemented afresh. But you do not change the tested piece of code in Savings and Checking account. This is advantage of OOP



- **Step 1)** Such that when the "withdrawn" method for saving account is called a method from parent account class is executed.


Account
deposit()
withdraw()
"Withdraw" action is directly performed from the parent class (Account)
withdraw()
saving
checking
privileged

- **Step 2)** But when the "Withdraw" method for the privileged account (overdraft facility) is called withdraw method defined in the privileged class is executed. This is **Polymorphism.**

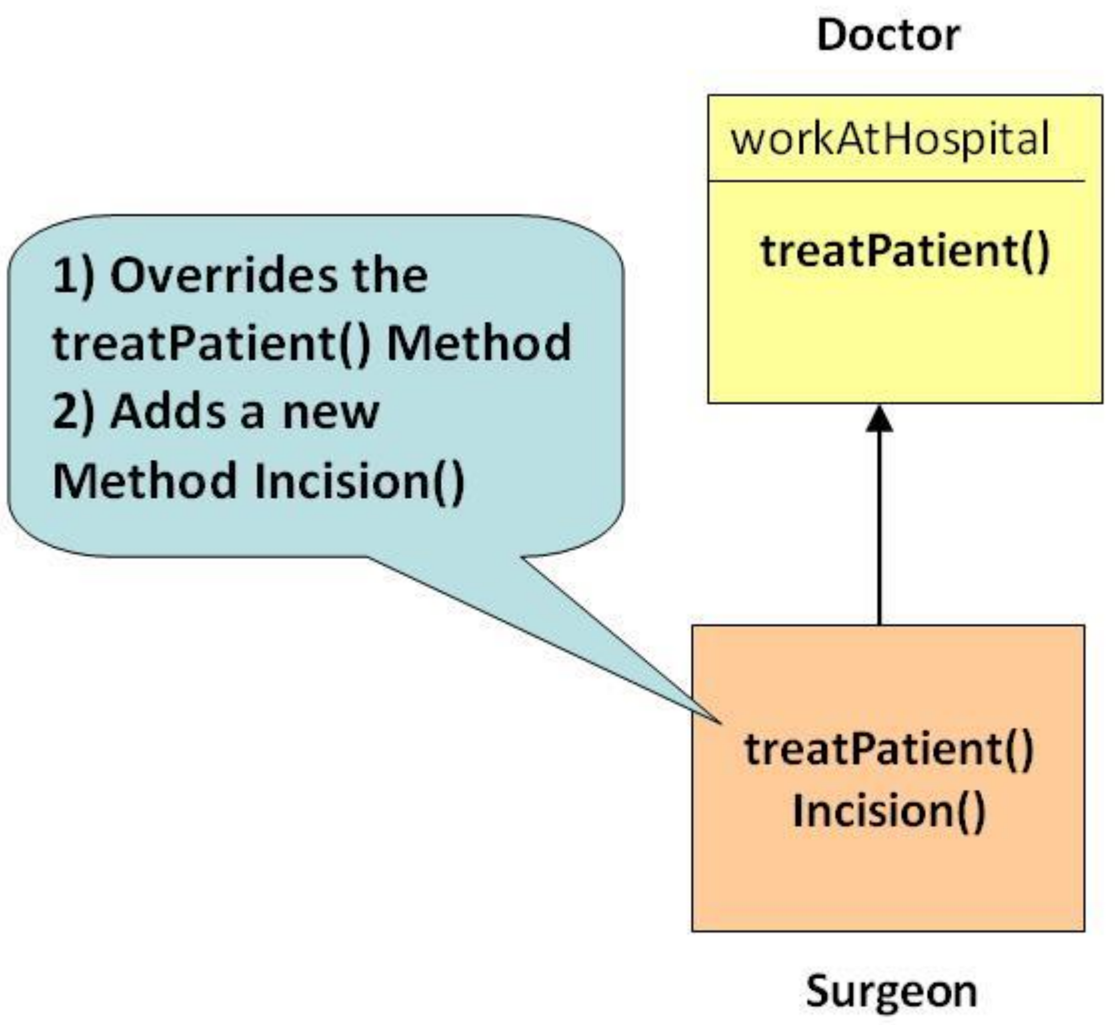

withdraw()
for privileged action (overdraft facility) custom "withdraw" action is accessed from class privileged
saving
checking
privileged

## Method Overriding

- Method Overriding is redefining a super class method in a sub class.

### Rules for Method Overriding

- The method signature i.e. method name, parameter list and return type have to match exactly.
- The overridden method can widen the accessibility but not narrow it, i.e. if it is private in the base class, the child class can make it public but not vice versa.


Doctor
workAtHospital
treatPatient()
1) Overrides the treatPatient() Method
2) Adds a new Method Incision()
treatPatient()
Incision()
Surgeon

SYNERGETICS GET IT RIGHT | Microsoft Partner Gold Cloud Platform Silver Learning

# Q & A

Contact: amitmahadik@synergetics-india.com

SYNERGETICS GET IT RIGHT

Microsoft Partner
Gold Cloud Platform
Silver Learning

# Thank You