Day7_OOPJ_Sanket_Shalukar

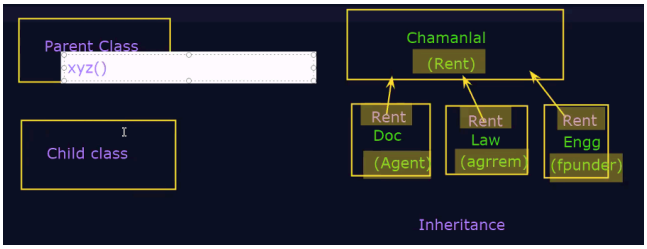
Wednesday, September 03, 2025 10:08 AM

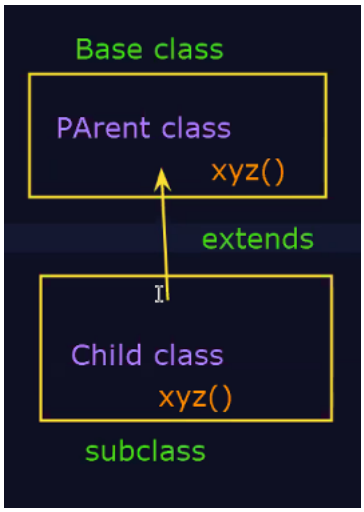
Topics are in the Day_7

- Abstraction
- Encapsulation
- Inheritance
- Polymorphism

- **Inheritance :**

The process where one class acquires the properties and behaviors (methods) of another class.
Example: A child inherits physical features and qualities from parents. Similarly, a Car class can inherit from a Vehicle class.





Superclass in Java

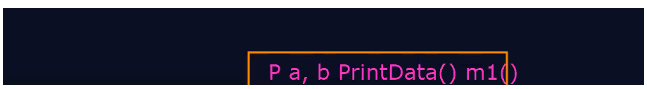
A **superclass** is the parent class from which another class (called a **subclass**) inherits fields and methods.

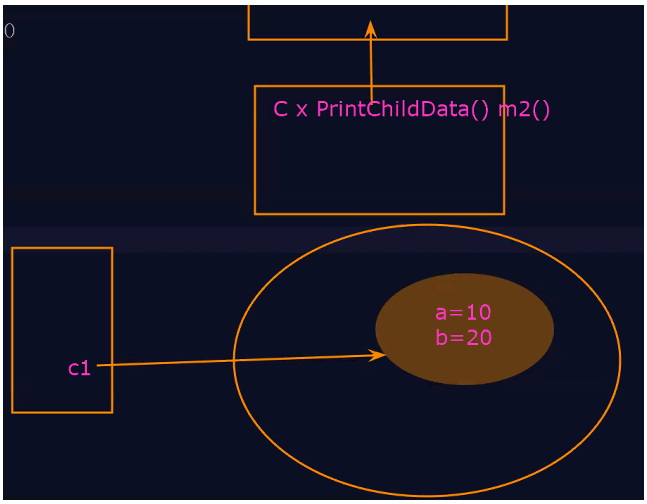
- The subclass uses the keyword **`extends`** to inherit from a superclass.
- Every class in Java implicitly extends the **`Object` class**, so `Object` is the ultimate superclass of all classes.
- Super class (Parent class) : THe Class whose properties are inherited.
- Sub class (Child Class) :The class that inherits properties from another class.

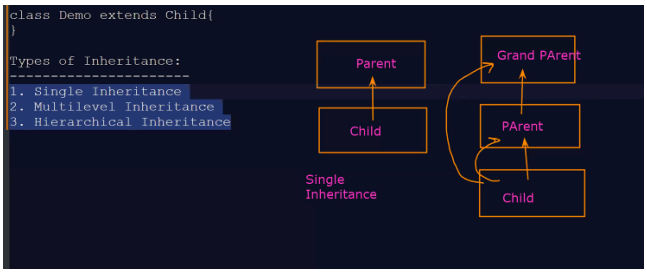
Reusability

Reusability is the ability to use existing code (classes, methods, modules) in new programs or contexts **without rewriting it**.

- It is a mechanism in java where a child class acquire properties (fields) and behavior (methods) from another class (parent class)







Types of Inheritance:

1. Single Inheritance
2. Multilevel Inheritance
3. Hierarchical Inheritance

- **Single Inheritance**

  When a class inherits from only **one superclass**, it is called single inheritance.

- **Multilevel Inheritance**

  When a class is derived from another subclass, forming a **chain of inheritance**, it is called multilevel inheritance.

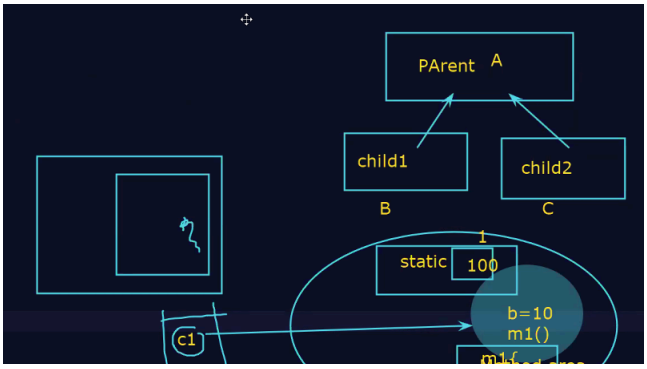
- **Hierarchical Inheritance**

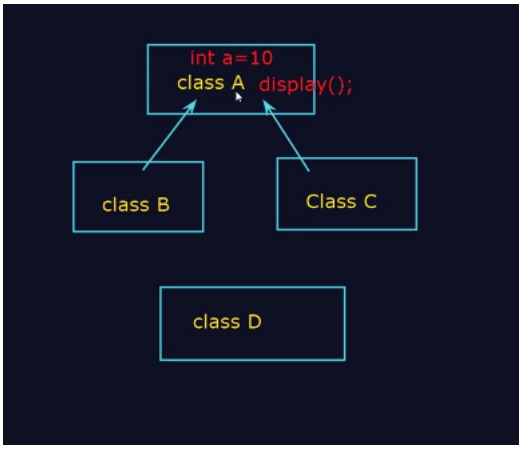
  When multiple classes inherit from the **same superclass**, it is called hierarchical inheritance.

Object creation:

1. Parent p = new Parent () ;
2. Parent p = null;
3. Parent p = new child() ;

Child c = new Parent () ;

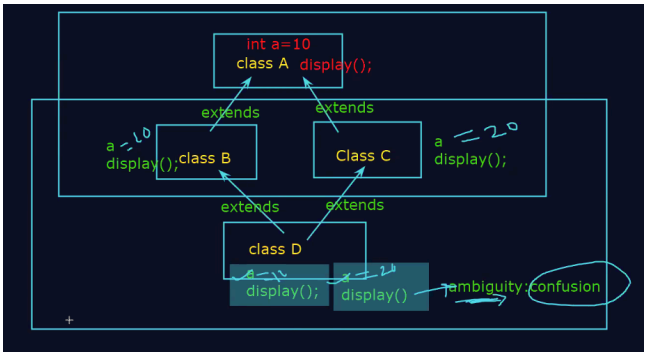




- **Ambiguity**

  Ambiguity means a situation where the compiler or program cannot decide which option to choose because of **confusion or multiple possibilities**.

- **Multiple Inheritance with Classes (Not Supported in Java)**
- If a class inherits from two classes having the same method, the compiler wouldn't know **which method to call**.
- That's why **Java does not support multiple inheritance with classes**.



Interface for Inheritance.

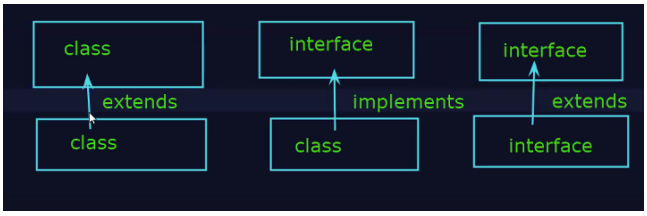
An **interface** in Java is a **blueprint of a class** that is used to achieve **100% abstraction** and **multiple inheritance**.
It can contain **abstract methods** (without body) and **constants** (public, static, final by default).

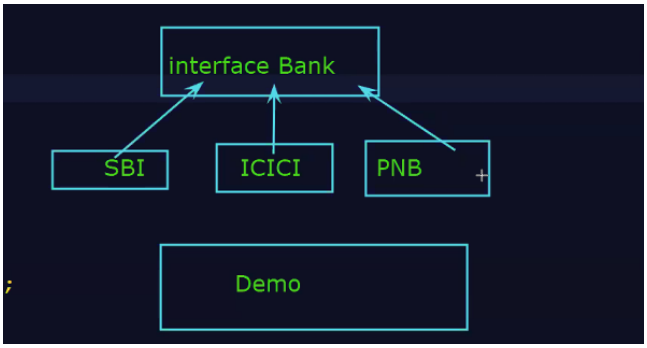
Key Points

- Interface = **blueprint for a class**.
- Contains **abstract methods and constants** by default (till Java 7).
- **Cannot have method body**, only declarations (till Java 7).
- Used to achieve **100% abstraction** and **multiple inheritance**.
- **Java 8** → Interface can have **default and static methods** (with body).
- **Java 9** → Interface can also have **private methods**.
- All variables in interface are **implicitly public, static, and final**.
- All methods are **implicitly public and abstract** (unless default, static, or private in Java 8+).

```
//constant variable
int x=100;
//abstract method
void read();
//default method: Java 8+
void show(){
SOP()
}
//static method : Java *+
static void display(){
```

```
    SOP();
}
//Private methods : Java 9+
private void print(){
}
```





- **Polymorphism :**

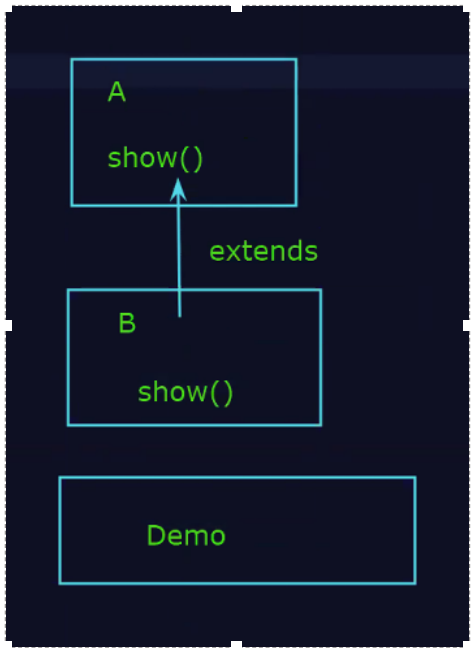
  The ability of a method or object to take many forms, allowing the same name to be used with different behaviors.
  Example: A person can be a student in school, a customer in a shop, and a player on the ground — same person, different roles **depending** on the situation.

  Poly=Many, Morphisam= Forms : many forms

- Types of polymorphism:

  1. Compile time polymorphism
     Method overloding!

  2. Run time polymorphism
     Method over riding!



- **Compile-Time Polymorphism (Method Overloading)**

When two or more methods in the same class have the **same name but different parameter lists** (different number or type of arguments).
The method call is resolved at **compile time** → so it's called **compile-time polymorphism**.

- **Run-Time Polymorphism (Method Overriding)**

  When a **subclass provides its own implementation** of a method that is already defined in its superclass.
  The method call is resolved at **runtime** → so it's called **runtime polymorphism**.