Day_8_OOPJ_Sanket Shalukar

Thursday, September 04, 2025 12:35 PM

Topics are in the Day_8

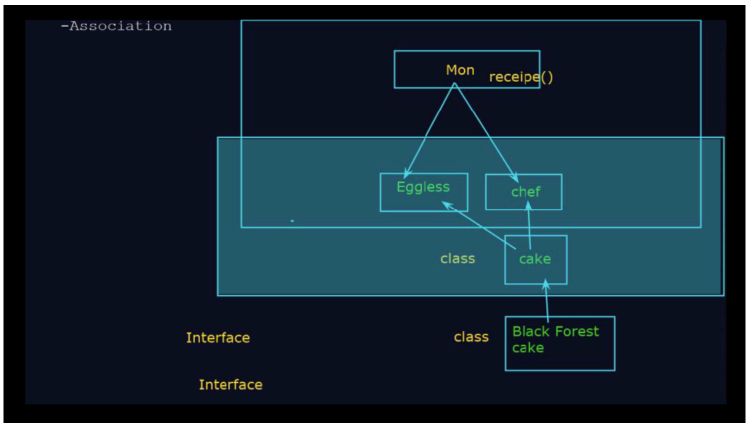
1. Inheritance
2. Polymorphism
3. Overriding
4. Super
5. Upcasting & downcasting
6. instanceof
7. Association

Inheritance

- Mechanism where a child class acquires properties and behavior from a parent class.
- Promotes **reusability**, **hierarchical organization**, and **polymorphism**.
- Represents an **IS-A** relationship.
- Example:
`class Parent { } class Child extends Parent { }`

Polymorphism

- Ability of the same method to perform different actions.
- **Compile-time (Overloading)** and **Runtime (Overriding)**.
- Core concept in **type hierarchy** and **upcasting/downcasting**.

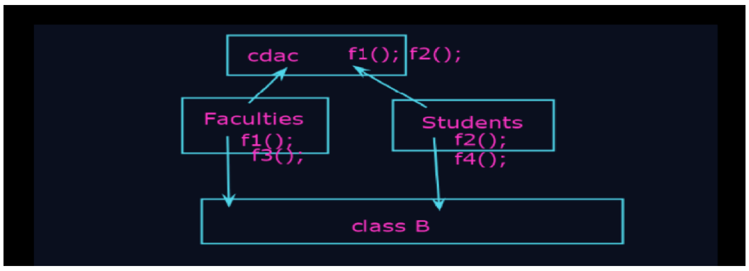


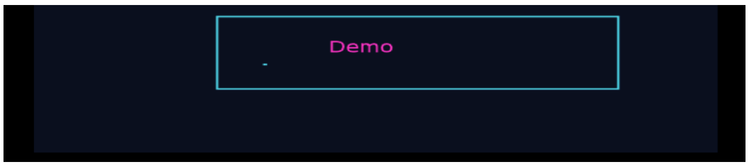
Method Overriding

- Child class provides its own implementation of a parent class method.
- Used to achieve **runtime polymorphism**.
- Rules: same method signature, same or covariant return type, access modifier not more restrictive.

super Keyword

- Used to access **parent class members and constructors**.
- Can call parent class constructor (`super()`), methods (`super.method()`), or variables (`super.var`).



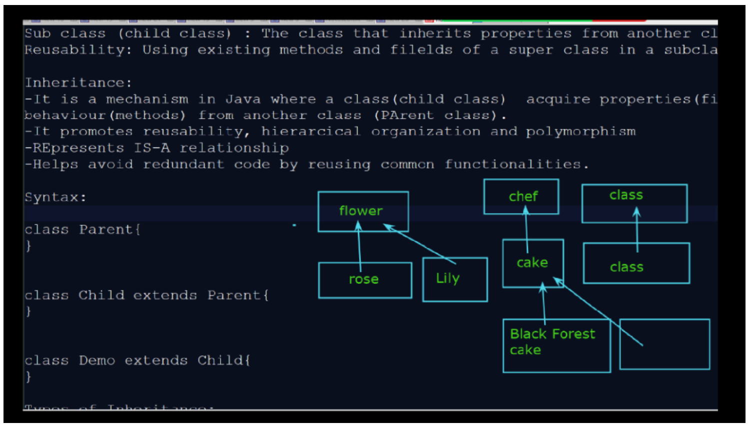


Upcasting & Downcasting!

Key concept used in polymorphism and type hierarchy.

Upcasting:

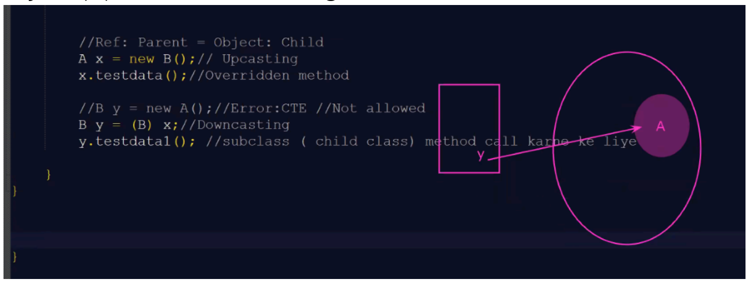
- Converting child class reference into parent class reference.
- Allows child class object to be treated as parent class object.
- Done implicitly, safe.
  `A x = new B(); // Upcasting`



```
Sub class (child class) : The class that inherits properties from another cl
Reusability: Using existing methods and filelds of a super class in a subcla

Inheritance:
-It is a mechanism in Java where a class(child class)  acquire properties(fi
behaviour(methods) from another class (PArent class).
-It promotes reusability, hierarcical organization and polymorphism
-REpresents IS-A relationship
-Helps avoid redundant code by reusing common functionalities.

Syntax:

class Parent{
}


class Child extends Parent{
}


class Demo extends Child{
}
```

Downcasting:

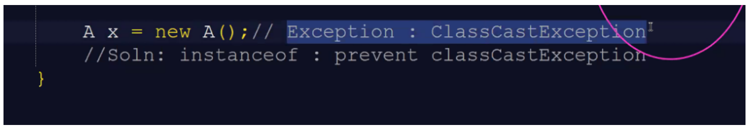
- Converting parent class reference into child class reference.
- Enables calling child-specific methods.
- Requires explicit cast, may throw `ClassCastException`.
  `B y = (B) x; // Downcasting`



```
        //Ref: Parent = Object: Child
        A x = new B();// Upcasting
        x.testdata();//Overridden method

        //B y = new A();//Error:CTE //Not allowed
        B y = (B) x;//Downcasting
        y.testdata1(); //subclass ( child class) method call karne ke liye

    }
}


}
```

instanceof Operator

- Tests whether an object is an instance of a class or subclass.
- Prevents invalid downcasting and `ClassCastException`.

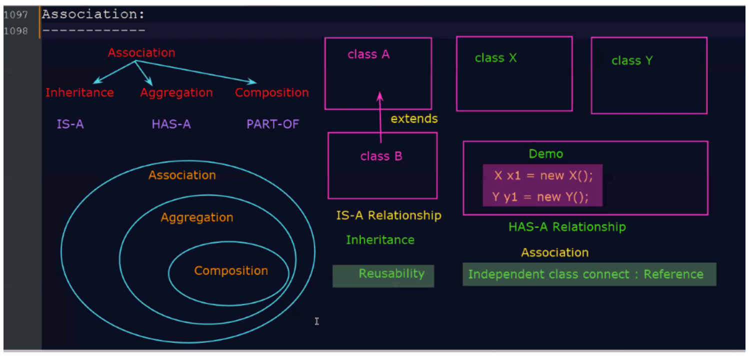


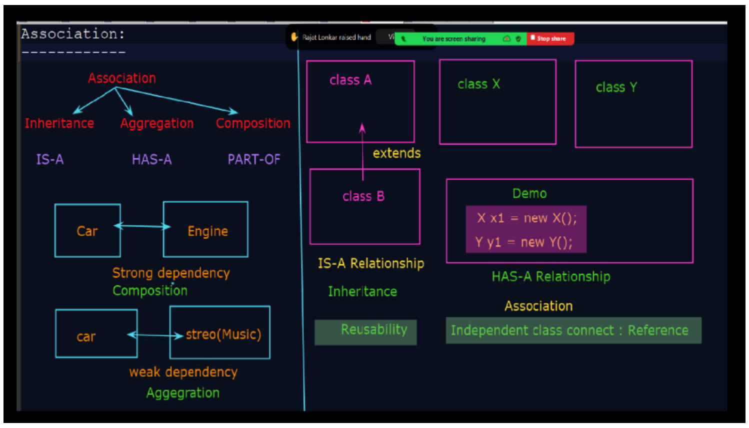
```
        A x = new A();// Exception : ClassCastException
        //Soln: instanceof : prevent classCastException
    }
```

Association

- Represents a relationship between two separate classes that are related but can exist independently.
- **Relationship between 2 classes ("uses-a").**
- **Types:**
- One-to-One
- One-to-Many

- Many-to-One
- Many-to-Many
- **Dependency:** Objects can exist independently.
- **Example:** Teacher–Student



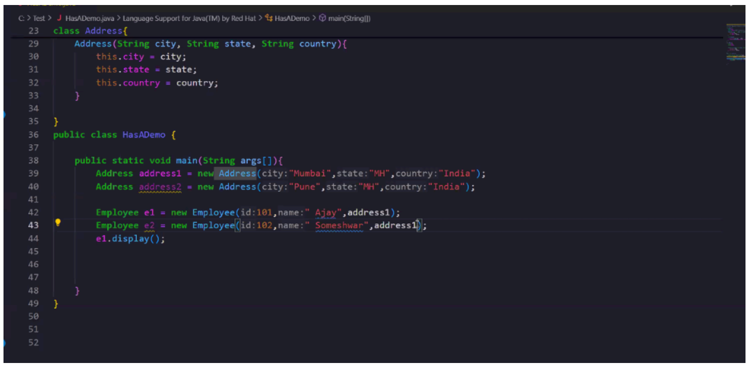


Aggregation

- One class has a reference to another class.
- **Weak Relationship (HAS-A).**
- Objects can exist independently.
- **Example:** Employee–Address

Composition

- One class owns another class.
- **Strong Relationship (PART-OF).**
- Contained objects cannot exist without container.
- If container object is destroyed, contained objects are also destroyed.
- **Example:** Car–Engine



C: > Test > HasADemo.java > Language Support for Java(TM) by Red Hat > HasADemo > main(String[])

```java
class Address{
    Address(String city, String state, String country){
        this.city = city;
        this.state = state;
        this.country = country;
    }

}
public class HasADemo {

    public static void main(String args[]){
        Address address1 = new Address(city:"Mumbai",state:"MH",country:"India");
        Address address2 = new Address(city:"Pune",state:"MH",country:"India");

        Employee e1 = new Employee(id:101,name:" Ajay",address1);
        Employee e2 = new Employee(id:102,name:" Someshwar",address1);
        e1.display();



    }
}
```