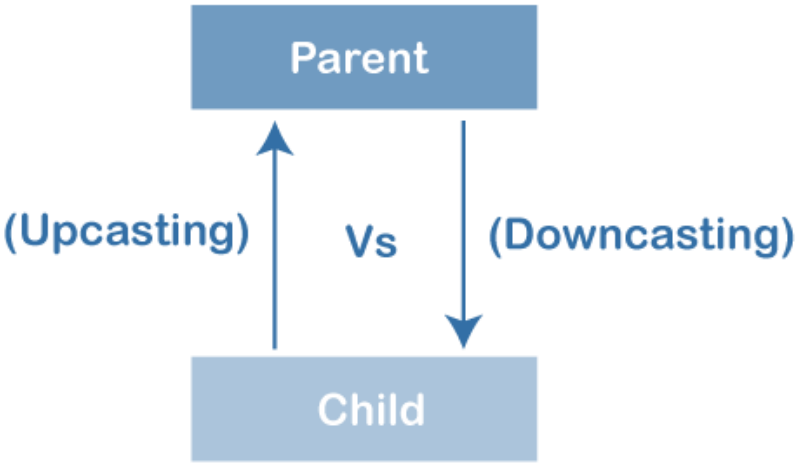
30-Upcasting and Downcasting

Typecasting is the process of converting one data type to another. In Java, this concept extends beyond basic data types to include objects, leading to what is known as **object typecasting**. Object typecasting involves converting a reference of one type into another, specifically between parent and child classes. This can be categorized into two types:

- **Upcasting** (Child to Parent)
- **Downcasting** (Parent to Child)

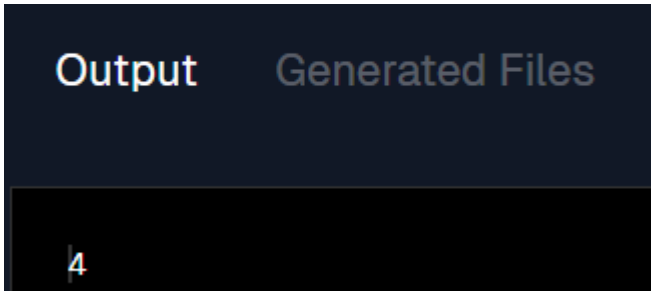


Typecasting ensures that variables are processed correctly by functions. While upcasting can be done implicitly or explicitly, downcasting requires explicit casting and can be risky if not handled properly.

Example of Typecasting

```java
public class Demo {
    public static void main(String[] args) {
        double d = 4.5;
        int i = (int) d;
        System.out.println(i); // Output: 4
    }
}
```

In this example, we are typecasting a double to an int. The compiler warns that data might be lost due to the conversion, but we accept it. The output is 4, demonstrating how typecasting works with primitive types.



Object-Oriented Typecasting (OOP)

Upcasting and **Downcasting** are essential concepts in Java's object-oriented programming (OOP). They allow us to cast objects from one type to another within an inheritance hierarchy.

Upcasting

Upcasting is the process of converting a child class reference to a parent class reference. It's a safe operation that doesn't require explicit casting because every child object is an instance of the parent class. Upcasting is also known as **generalization** or **widening**.

- **Benefits of Upcasting**:
    - Simplifies code by allowing a uniform interface to handle different objects.
    - Enables polymorphism, where a single method can be used on objects of different types.

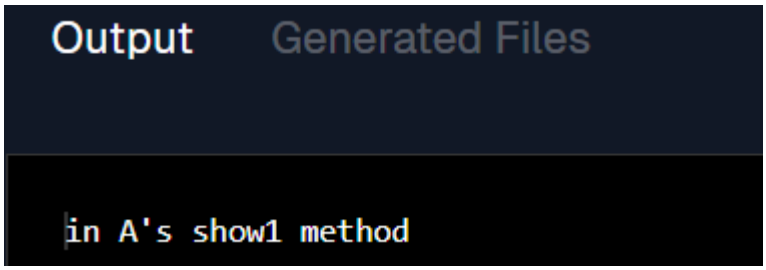
Example:

```java
class A {
    public void show1() {
        System.out.println("in A's show1 method");
    }
}

class B extends A {
    public void show2() {
        System.out.println("in B's show2 method");
    }
}

public class Demo {
    public static void main(String[] args) {
        A obj = new B(); // Upcasting
        obj.show1(); // Output: in A's show1 method
    }
}
```

In this example, B is upcasted to A. The show1() method of class A is accessible, but the show2() method of B is not, since the reference type is A.



Downcasting

Downcasting is the reverse of upcasting, where we cast a parent class reference back to a child class reference. This operation is more restrictive and can lead to runtime exceptions if not handled correctly. In Java, downcasting must be done explicitly using the cast operator.

- **Risks of Downcasting**:
    - Can throw ClassCastException if the object being cast is not actually an instance of the target class.
    - Should be used cautiously, often with checks using the instanceof operator.

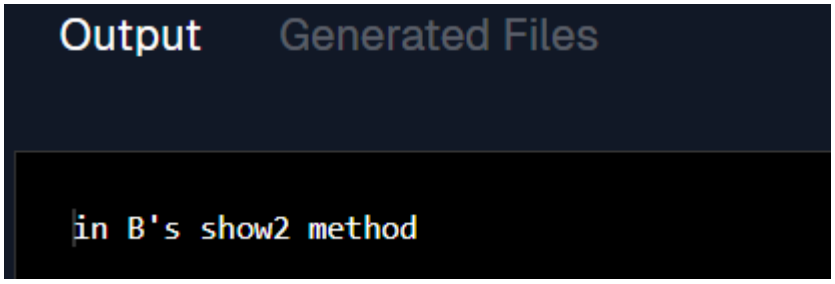
Example:

```java
class A {
    public void show1() {
        System.out.println("in A's show1 method");
    }
}

class B extends A {
    public void show2() {
        System.out.println("in B's show2 method");
    }
}

public class Demo {
    public static void main(String[] args) {
        A obj = new B(); // Upcasting
        B obj1 = (B) obj; // Downcasting
        obj1.show2(); // Output: in B's show2 method
    }
}
```

Here, obj is first upcasted to A, then downcasted back to B. The method show2() from class B is accessible after downcasting.



Conclusion

Whether you are performing upcasting or downcasting, these operations are fundamental to understanding polymorphism and inheritance in Java. Upcasting is generally safer and more commonly used, while downcasting requires careful handling to avoid runtime errors.

FAQs on Upcasting and Downcasting

1. **What is the difference between upcasting and downcasting?**
    - Upcasting refers to converting a child class reference to a parent class reference, while downcasting is converting a parent class reference back to a child class reference.
2. **Is upcasting always safe in Java?**
    - Yes, upcasting is safe and can be done implicitly because every child class object is an instance of its parent class.
3. **Why is downcasting not safe?**
    - Downcasting can lead to ***ClassCastException*** at runtime if the object being cast is not an instance of the target class. It requires explicit casting and often involves runtime checks using the instanceof operator.
4. **Can we access child class methods after upcasting?**
    - No, after upcasting, only methods and variables defined in the parent class are accessible. To access child class-specific methods, downcasting is required.

.