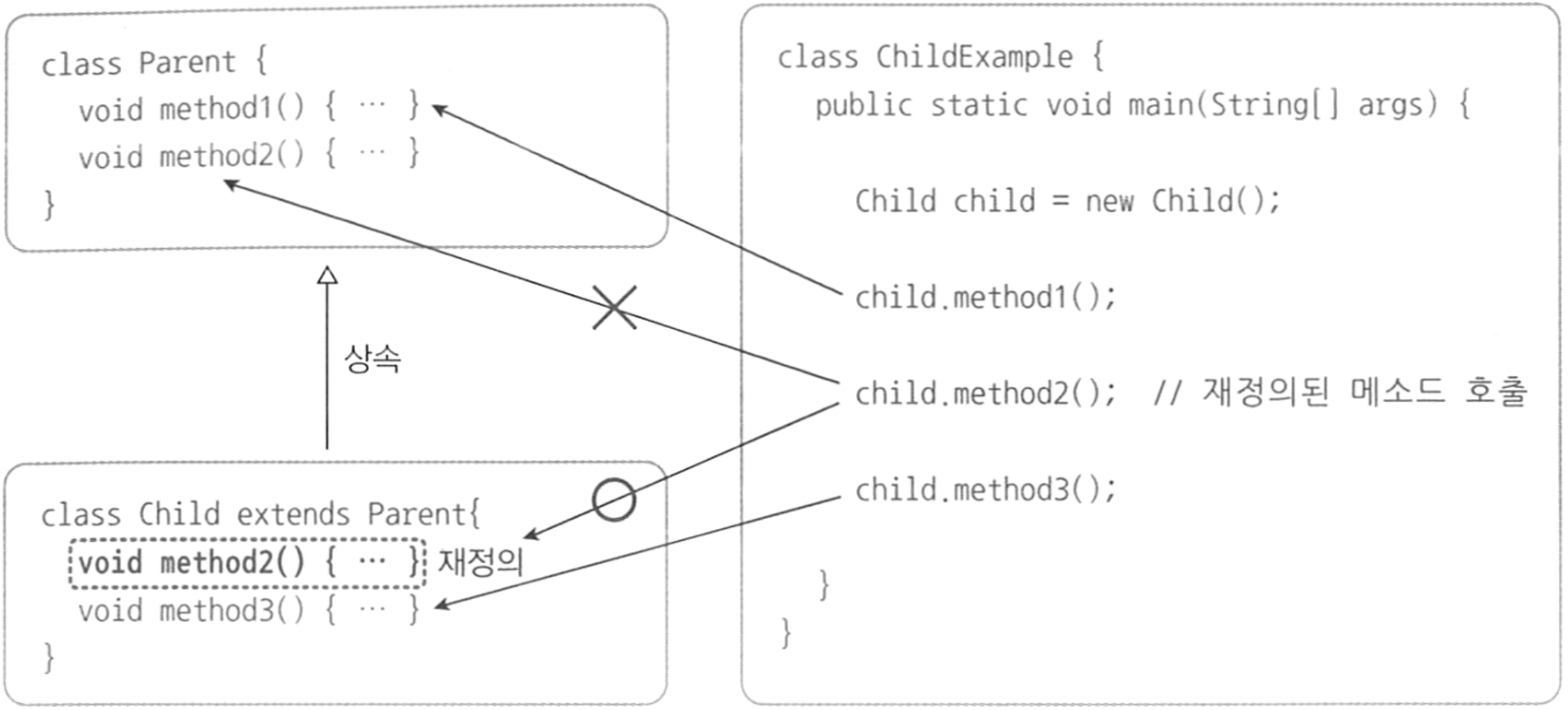
메소드 재정의(@Override)

메소드 재정의(@Override)

❖ 메소드 재정의(@Override)

- ㅇ 부모 클래스의 상속 메소드 수정해 자식 클래스에서 재정의하는 것



메소드 재정의(@Override)

❖ 메소드 재정의 조건

- 부모 클래스의 메소드와 동일한 시그니처 가져야
- 접근 제한을 더 강하게 오버라이딩 불가
  - public을 default나 private으로 수정 불가
  - 반대로 default는 public 으로 수정 가능

- 새로운 예외(Exception) throws 불가 (예외처리는 10장 참조)

메소드 재정의(@Override)

❖ 부모 클래스: Calculator.java

```
public class Calculator {
    double areaCircle(double r) {
        System.out.println("Calculator 객체의 areaCircle() 실행");
        return 3.14159 * r * r;
    }
}
```

❖ 자식 클래스 : Computer.java

```
public class Computer extends Calculator {
    @Override
    double areaCircle(double r) {
        System.out.println("Computer 객체의 areaCircle() 실행");
        return Math.PI * r * r;
    }
}
```

메소드 재정의(@Override)

❖ 자식 클래스 사용: ComputerExample.java

```
public class ComputerExample {
    public static void main(String[] args) {
        int r = 10;

        Calculator calculator = new Calculator();
        System.out.println("원면적 : " + calculator.areaCircle(r));
        System.out.println();

        Computer computer = new Computer();

        // 재정의된 메서드 호출
        System.out.println("원면적 : " + computer.areaCircle(r));
    }
}
```

메소드 재정의(@Override)

❖ @Override 어노테이션

- 컴파일러에게 부모 클래스의 메소드 선언부와 동일한지 검사 지시

- 정확한 메소드 재정의 위해 붙여주면 OK

❖ 메소드 재정의 효과

- 부모 메소드는 숨겨지는 효과 발생
  - 재정의된 자식 메소드 실행

메소드 재정의(@Override)

❖ 부모 메소드 사용(super)

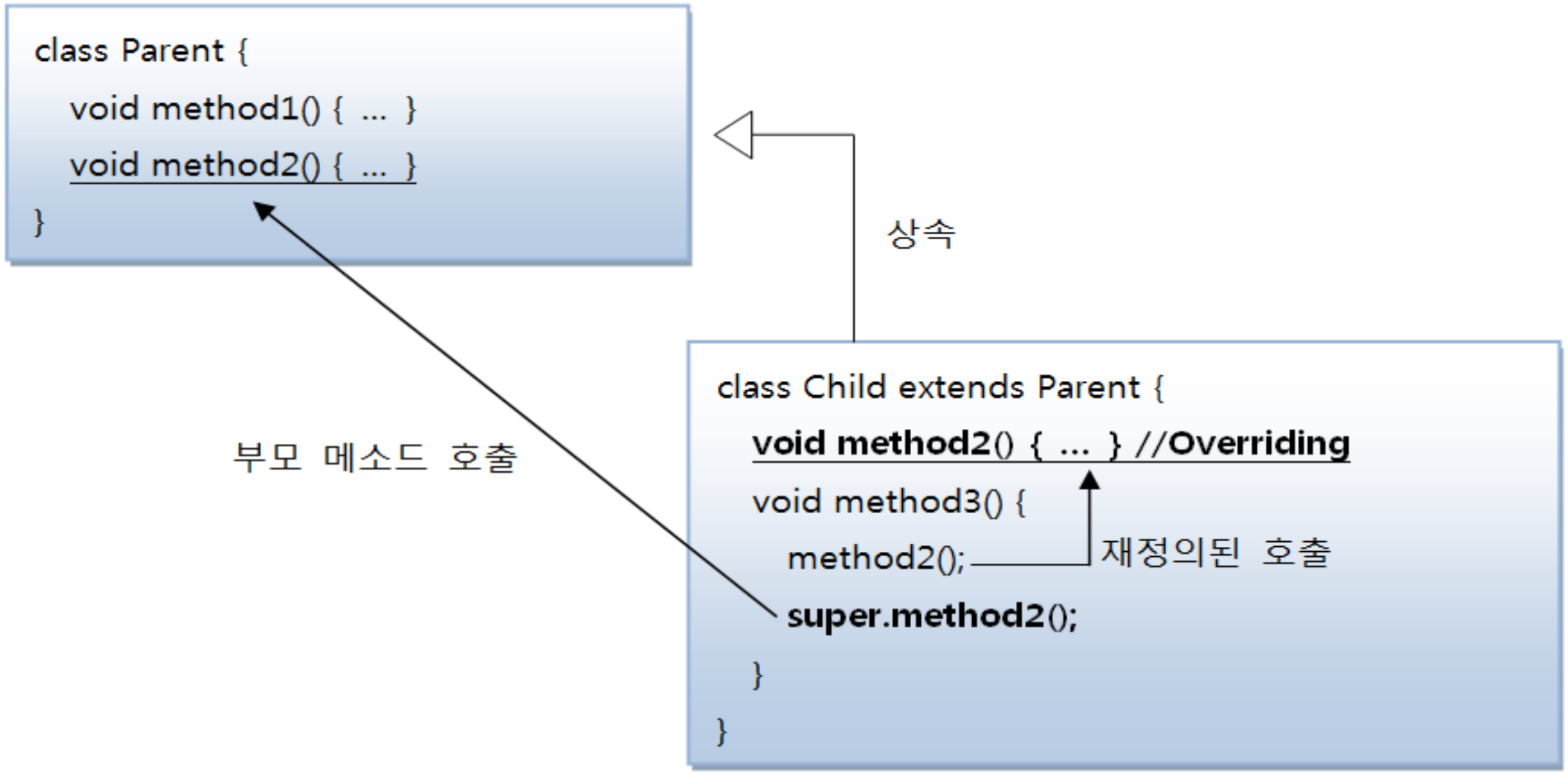
- 메소드 재정의는 부모 메소드 숨기는 효과 !!
  - 자식 클래스에서는 재정의된 메소드만 호출

- 자식 클래스에서 수정되기 전 부모 메소드 호출 - super 사용
  - super는 부모 객체 참조(참고: this는 자신 객체 참조)

메소드 재정의(@Override)

❖ 부모 메소드 사용(super)

super.부모메소드();



메소드 재정의(@Override)

❖ super 변수: Airplane.java

```
public class Airplane {
    public void land() {
        System.out.println("착륙합니다.");
    }

    public void fly() {
        System.out.println("일반비행합니다.");
    }

    public void takeOff() {
        System.out.println("이륙합니다.");
    }
}
```

메소드 재정의(@Override)

❖ super 변수: SupersonicAirplane.java

```
public class SupersonicAirplane extends Airplane {
    public static final int NORMAL = 1;
    public static final int SUPERSONIC = 2;

    public int flyMode = NORMAL;

    @Override
    public void fly() {
        if (flyMode == SUPERSONIC) {
            System.out.println("초음속비행합니다.");
        } else {
            // Airplane 객체의 fly() 메소드 호출
            super.fly();
        }
    }
}
```

메소드 재정의(@Override)

❖ super 변수: SupersonicAirplaneExample.java

```
public class SupersonicAirplaneExample {
    public static void main(String[] args) {
        SupersonicAirplane sa = new SupersonicAirplane();
        sa.takeOff();
        sa.fly();

        sa.flyMode = SupersonicAirplane.SUPERSONIC;
        sa.fly();

        sa.flyMode = SupersonicAirplane.NORMAL;
        sa.fly();

        sa.land();
    }
}
```

final 클래스와 final 메소드

❖ final 키워드의 용도

- final 필드: 수정 불가 필드
- final 클래스: 부모로 사용 불가한 클래스
- final 메소드: 자식이 재정의할 수 없는 메소드

❖ 상속할 수 없는 final 클래스

- 자식 클래스 만들지 못하도록 final 클래스로 생성

```
public final class 클래스 { ... }
```

```
public final class String { .. }
```

```
public class NewString extends String { ... }
```

❖ 오버라이딩 불가한 final 메소드

- 자식 클래스가 재정의 못하도록 부모 클래스의 메소드를 final로 생성