■ 메소드 재정의 (Overriding)

"조상클래스로부터 상속받은 메서드의 내용을 상속받는 클래스에 맞게 변경하는 것을 오버라이딩이라고 한다."

```
class Point {
      int x;
      int y;

      String getLocation() {
            return "x :" + x + ", y :"+ y;
      }
}

class Point3D extends Point {
      int z;
      String getLocation() {        // 오버라이딩
            return "x :" + x + ", y :"+ y + ", z :" + z;
      }
}
```

■ 메소드 재정의 (Overriding) 의 조건

- 선언부가 같아야 한다.(이름, 매개변수, 리턴타입)
- 접근제어자를 좁은 범위로 변경할 수 없다.

  - public ➔ default 또는 private 변경 불가

  - default ➔ public 또는 protected 변경 가능
- 조상클래스의 메서드보다 많은 수의 예외를 선언할 수 없다.

```
class Parent {
    void parentMethod() throws IOException, SQLException {
        // ...
    }
}

class Child extends Parent {
    void parentMethod() throws IOException {
        //..
    }
}

class Child2 extends Parent {
    void parentMethod() throws Exception {
        //..
    }
}
```

■ @Override 애노테이션 사용

- 컴파일러에게 부모 클래스의 메소드 선언부와 동일한지 검사 지시

- 정확한 메소드 재정의를 위해 붙여 주는 것을 권장

■ 메소드 재정의 효과

- 부모 메소드는 숨겨지는 효과 발생

- 재정의된 자식 메소드 실행

■ Overriding (1 / 4)

```
public class Car {
    int tire = 4;
    int door = 2;

    Car() {
        System.out.println("Car 객체 생성");
    }

    void move() {
        System.out.println("이동");
    }

    public int getTire() {
        return tire;
    }

    public void setTire(int tire) {
        this.tire = tire;
    }

    public int getDoor() {
        return door;
    }

    public void setDoor(int door) {
        this.door = door;
    }
}
```

■ Overriding (2 / 4)

```
public class SportsCar extends Car {
    public SportsCar() {
        System.out.println("SportsCar 객체 생성");
    }

    void openSunloof() {
        System.out.println("썬루프 열림");
    }

    @Override
    void move() {
        System.out.println("100km/h 이동");
    }
}
```

■ Overriding (3 / 4)

```
public class Truck extends Car {
    public Truck() {
        System.out.println("Truck 객체 생성");
    }

    void load() {
        System.out.println("짐 실음");
    }

    @Override
    void move() {
        System.out.println("50km/h 이동");
    }
}
```

■ Overriding (4 / 4)

```
public class CarMain {
    public static void main(String[] args) {
        Car car = new Car();
        car.move();

        SportsCar sCar = new SportsCar();
        sCar.move();

        Truck tCar = new Truck();
        tCar.move();

        Car car2 = new Truck();
        car2.move();

        Car car3 = new SportsCar();
        car3.move();
    }
}
```

```
Car 객체 생성
이동
```

```
Car 객체 생성
SportsCar 객체 생성
100km/h 이동
```

```
Car 객체 생성
Truck 객체 생성
50km/h 이동
```

```
Car 객체 생성
Truck 객체 생성
50km/h 이동
```

```
Car 객체 생성
SportsCar 객체 생성
100km/h 이동
```

■ 오버로딩 (Overloading)

- 클래스 내에 같은 이름의 메소드를 여러 개 선언하는 것
- 하나의 메소드 이름으로 다양한 매개값 받기 위해 메소드 오버로딩
- 오버로딩의 조건: 매개변수의 타입, 개수, 순서가 달라야 됨

■ 오버로딩의 조건

- 메서드의 이름이 같아야 한다.
- 매개변수의 개수 또는 타입이 달라야 한다. (리턴 타입은 상관없음)
- 매개변수는 같고 리턴타입이 다른 경우는 오버로딩이 성립되지 않는다.

  ※ 오버로딩된 메소드는 호출시 전달하는 인자를 통해서 구분

■ 오버로딩의 조건

```
class 클래스 {
   리턴타입  메소드이름 ( 타입 변수, ... )  { ... }
```

무관 / 동일 / 매개변수의 타입, 개수, 순서가 달라야함

```
   리턴타입  메소드이름 ( 타입 변수, ... )  { ... }
}
```

```
void println() { .. }
void println(boolean x) { .. }
void println(char x) { .. }
void println(char[] x) { .. }
void println(double x) { .. }
void println(float x) { .. }
void println(int x) { .. }
void println(long x) { .. }
void println(Object x) { .. }
void println(String x) { .. }
```

```
int plus(int x, int y) {
   int result = x + y;
   return result;
}
```

```
plus(10, 20);
```

```
double plus(double x, double y) {
   double result = x + y;
   return result;
}
```

```
plus(10.5, 20.3);
```

```
int x = 10;
double y = 20.3;
plus(x, y);
```

?

```
int divide(int x, int y) { ... }
double divide(int boonja, int boonmo) { ... }
```

■ Overloading

```
public class OverloadingExam {
    void change(int num) {}

    void change(char ch) {}

    void change(int num, char ch) {}

//    int change(int num) {} 반환 타입만 다른 경우 오류, 동일 메소드로 인식
}
```

형변환의 규칙까지 적용해야만 메소드가 구분되는 기막히게 애매한 상황은 만들지 말자!

```
class AAA
{
    void isYourFunc(int n) { . . . }
    void isYourFunc(int n1, int n2) { . . . }
    void isYourFunc(int n1, double n2) { . . . }
     . . . .
}
```

오버로딩 된 메소드

```
AAA inst = new AAA();
inst.isYourFunc(10, 'a');
```

무엇이 호출되겠는가? 문자 'a'는 int형으로도, double형으로도 변환이 가능하다!

결론적으로, 형변환 규칙을 적용하되 가장 가까운 위치의 자료형으로 변환이 이뤄진다. 따라서 is...(int n1, int n2)가 호출된다.

■ 오버로딩 vs 오버라이딩

오버로딩(overloading) - 기존에 없는 새로운 메서드를 정의하는 것(new)
오버라이딩(overriding) - 상속받은 메서드의 내용을 변경하는 것(change, modify)

```
class Parent {
    void parentMethod() {}
}

class Child extends Parent {
    void parentMethod() {}          // 오버라이딩
    void parentMethod(int i) {}     // 오버로딩

    void childMethod() {}
    void childMethod(int i) {}      // 오버로딩
    void childMethod() {}           // 에러!!! 중복정의임
}
```

■ super / this 사용 – 1 (변수)

```
public class Parent {
    int num = 10;
}
```

```
public class Child extends Parent {
    int num = 20;

    void show(int num) {
        System.out.println("지역변수 : " + num);
        System.out.println("전역변수 : " + this.num);
        System.out.println("부모 전역변수 : " + super.num);
    }
}
```

```
public class SuperThis1 {
    public static void main(String[] args) {
        Child c = new Child();
        c.show(30);
    }
}
```

■ super / this 사용 – 2 (메소드)

```
public class Parent {
    int x;
    int y;

    public String getLocation() {
        return "x : " + x + ", y : " + y;
    }
}
```

```
public class Child extends Parent {
    int z;

    @Override
    public String getLocation() {
//        return "x : " + x + ", y : " + y + ", z : " + z;
        return super.getLocation() + ", z : " + z;
    }
}
```

```
public class SuperThis2 {
    public static void main(String[] args) {
        Child c = new Child();
        System.out.println(c.getLocation());
    }
}
```