

# 클래스, 객체

상속, Object, super, super(), 오버라이딩, 상속관계 객체참조, type casting, instanceof

# 클래스 작성: 상속

다음 두 클래스의 정의를 비교하시오

```
public class Robot {  
    String id;  
    public Robot(String id) {  
        this.id=id;  
    }  
    @Override  
    public String toString() {  
        return id;  
    }  
}
```

```
public class HumanoidRobot {  
    String id; // 제조회사에서 부여된 식별번호  
    String address; // 로봇의 주소  
    String name; // 로봇에게 부여된 이름  
    public HumanoidRobot(String id, String address, String name) {  
        this.id=id;  
        this.address=address;  
        this.name=name;  
    }  
    @Override  
    public String toString() {  
        return id+","+address+","+name;  
    }  
}
```

# 클래스 작성: 상속

## 클래스 상속(inheritance)

- 이미 작성된 Robot 클래스가 존재한다고 가정할 때
- Robot 클래스의 정의를 재사용(확장, extends)하여 HumanoidRobot 클래스를 정의

```
public class Robot{  
    String id;  
    public Robot(String id) {  
        this.id=id;  
    }  
    @Override  
    public String toString() {  
        return id;  
    }  
}
```

```
public class HumanoidRobot extends Robot{  
    String address;  
    String name;  
    public HumanoidRobot(String id, String address, String name) {  
        super(id);  
        this.address=address;  
        this.name=name;  
    }  
    @Override  
    public String toString() {  
        return super.toString()+","+address+","+name;  
    }  
}
```

# 클래스 작성: 상속, super, super(), 오버라이딩



상속(inheritance): A extends B

- 기존 클래스 B를 확장(extend)하는 방식으로 새로운 클래스 A를 정의

- ◆ A extends B

- B → 부모 클래스(parent class), 상위 클래스(super-class)
- A → 자식 클래스(child class), 하위 클래스(sub-class)

- ◆ super() → 부모 클래스의 생성자

- ◆ super → 부모 클래스의 객체 참조

- 오버라이딩(overriding)

- ◆ 아래 HumanoidRobot 클래스 정의문의 toString() 은 부모클래스 Robot의 toString()을 HumanoidRobot 클래스에 맞게 고쳐 쓴 것 (즉 overwriting 혹은 overriding한 것)

```
public class Robot{  
    String id;  
    public Robot(String id) {  
        this.id=id;  
    }  
    @Override  
    public String toString() {  
        return id;  
    }  
}
```

```
public class HumanoidRobot extends Robot{  
    String address;  
    String name;  
    public HumanoidRobot(String id, String address, String name) {  
        super(id);  
        this.address=address;  
        this.name=name;  
    }  
    @Override  
    public String toString() {  
        return super.toString()+","+address+","+name;  
    }  
}
```

← 부모 클래스 생성자 호출문은 자식 클래스 생성자 내 첫 행에 위치  
참조: <https://docs.oracle.com/javase/tutorial/java/landl/super.html>

# 오버라이딩 vs. 오버로딩

## 오버라이딩(overriding)

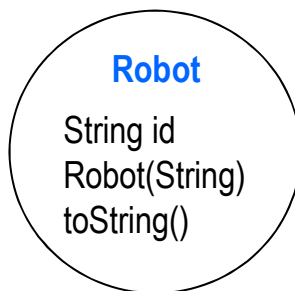
- 상위 클래스 내 메소드와 파라미터 개수, 파라미터 타입 및 리턴 타입이 모두 같은 메소드를 하위 클래스 내에 정의하는 것

## 오버로딩(overloading)

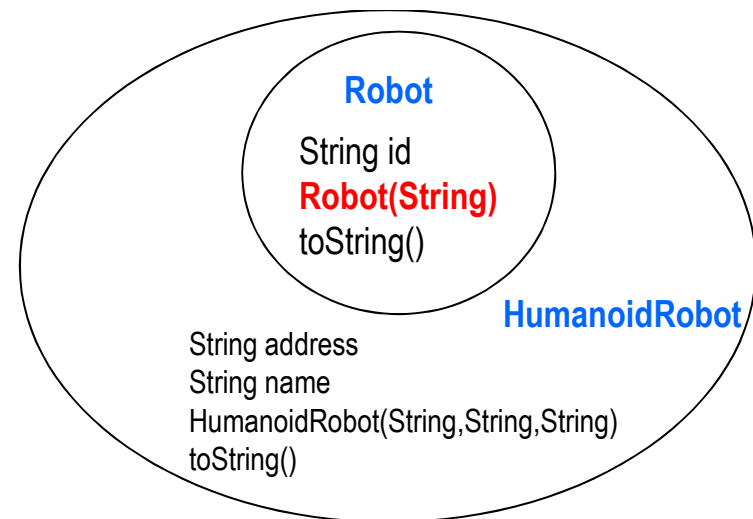
- 파라미터의 개수나 타입이 다르면서 이름이 같은 생성자 혹은 메소드를 정의하는 것 (리턴 타입만 다른 경우는 오버로딩 아님)

# 클래스 작성: 상속과 포함관계

```
public class Robot {  
    String id;  
    public Robot(String id) {  
        this.id=id;  
    }  
    @Override  
    public String toString() {  
        return id;  
    }  
}
```



```
public class HumanoidRobot extends Robot {  
    String address;  
    String name;  
    public HumanoidRobot(String id, String address, String name) {  
        super(id);  
        this.address=address;  
        this.name=name;  
    }  
    @Override  
    public String toString() {  
        return super.toString()+","+address+","+name;  
    }  
}
```

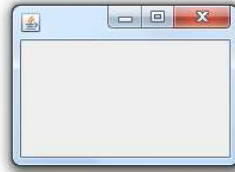


출처: <https://docs.oracle.com/javase/tutorial/java/landl/subclasses.html>

- 하위클래스는 상위클래스의 접근 가능한 모든 member들을 상속
- 생성자는 member가 아니므로 상속되는 것은 아니며 하위클래스로부터 `super()`를 통해 호출 가능

# 클래스 상속 예: MyWindow extends JFrame

```
public class Test {  
    public static void main(String[] args) {  
        JFrame w=new JFrame();  
        w.setVisible(true);  
    }  
}
```

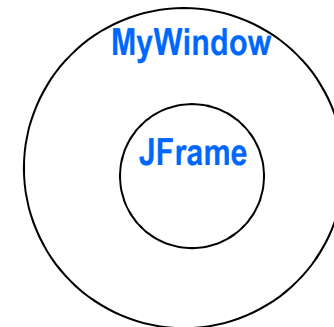
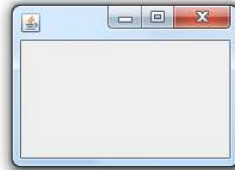


---

JFrame 클래스를 상속 받아 새로운 MyWindow 클래스를 정의

```
public class MyWindow extends JFrame {  
}
```

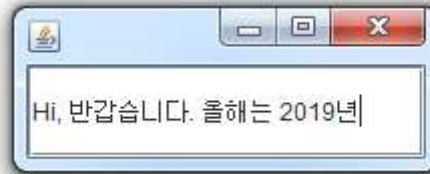
```
public class Test {  
    public static void main(String[] args) {  
        MyWindow w=new MyWindow();  
        w.setVisible(true);  
    }  
}
```



JFrame  
|  
MyWindow

# 클래스 상속 예: NumberTextField extends JTextField

```
public class Test {  
    public static void main(String[] args) {  
        JFrame w=new JFrame();  
        JTextField textField=new JTextField();  
        w.add(textField);  
        w.pack();  
        w.setVisible(true);  
    }  
}
```



JTextField 객체를 통해  
한글, 영문, 숫자 등 입력 가능

```
public class NumberTextField extends JTextField {  
    @Override  
    public void replaceSelection(String content) {  
        super.replaceSelection(content.replaceAll("[^0-9]", ""));  
    }  
}
```

JTextField 클래스를 상속받아 숫자만 입력 가능한  
새로운 NumberTextField 클래스 정의 (참조:  
<https://docs.oracle.com/javase/tutorial/java/javaOO/anonymousclasses.html>)

```
public class Test {  
    public static void main(String[] args) {  
        JFrame w=new JFrame();  
        NumberTextField textField=new NumberTextField();  
        w.add(textField);  
        w.pack();  
        w.setVisible(true);  
    }  
}
```



NumberTextField 객체를 통해  
숫자만 입력 가능



# 클래스 Object

## 클래스 Object

- Object를 제외한 다른 모든 자바 클래스의 부모 혹은 조상 클래스
- 클래스 Object는 부모 클래스가 없음
- 부모 클래스 없이 정의된 클래스는 암묵적으로 Object의 자식 클래스

```
public class Person {  
  
}
```

```
public class Person extends Object {  
  
}
```

# 상속 클래스 정의: super()

```
public class Person {  
    String name;  
    public Person() {  
        System.out.println("Person 객체 생성");  
    }  
}
```

```
public class Student extends Person {  
    int score;  
    public Student() {  
        System.out.println("Student 객체 생성");  
    }  
}
```

```
public class Test {  
    public static void main(String[] args) {  
        Students=new Student();  
    }  
}
```

실행결과:  
Person 객체 생성  
Student 객체 생성

```
public class Person {  
    String name;  
    public Person() {  
        System.out.println("Person 객체 생성");  
    }  
}
```

```
public class Student extends Person {  
    int score;  
    public Student() {  
        super();  
        System.out.println("Student 객체 생성");  
    }  
}
```

부모클래스 생성자 호출문 누락 시  
부모클래스 기본생성자 호출문  
super(); 자동 삽입

```
public class Test {  
    public static void main(String[] args) {  
        Students=new Student();  
    }  
}
```

```
public class Person extends Object {  
    String name;  
    public Person() {  
        super();  
        System.out.println("Person 객체 생성");  
    }  
}
```

```
public class Student extends Person {  
    int score;  
    public Student() {  
        super();  
        System.out.println("Student 객체 생성");  
    }  
}
```

```
public class Test {  
    public static void main(String[] args) {  
        Students=new Student();  
    }  
}
```

# 상속 클래스 정의: super()

```
public class Person {  
    String name;  
    public Person(String name) {  
        this.name=name;  
    }  
}
```

```
public class Student extends Person {  
    int score;  
    public Student(String name, int score) {  
        this.name=name;  
        this.score=score;  
    }  
}
```

- 오류 발생

```
public class Test {  
    public static void main(String[] args) {  
        Students=new Student("홍길동", 98);  
    }  
}
```

```
public class Person {  
    String name;  
    public Person(String name) {  
        this.name=name;  
    }  
}
```

Person 클래스 내 생성자  
존재하므로 기본 생성자  
자동 삽입되지 않음

```
public class Student extends Person {  
    int score;  
    public Student(String name, int score) {  
        super();  
        this.name=name;  
        this.score=score;  
    }  
}
```

- 오류 발생  
부모클래스 생성자 호출문 누락 시  
부모클래스 기본생성자 호출문 super();  
자동 삽입되나, 부모클래스 Person에는  
기본 생성자 부재

```
public class Test {  
    public static void main(String[] args) {  
        Students=new Student("홍길동", 98);  
    }  
}
```

```
public class Person {  
    String name;  
    public Person(String name) {  
        this.name=name;  
    }  
    public Person() {  
    }  
}
```

```
public class Student extends Person {  
    int score;  
    public Student(String name, int score) {  
        super();  
        this.name=name;  
        this.score=score;  
    }  
}
```

```
public class Test {  
    public static void main(String[] args) {  
        Students=new Student("홍길동", 98);  
    }  
}
```

## 상속 클래스 정의: super()

```
public class Person {  
}
```

```
public class Student extends Person {  
}
```

```
public class Test {  
    public static void main(String[] args) {  
        Student s=new Student();  
    }  
}
```

```
public class Person {  
    public Person() {  
        super();  
    }  
}
```

```
public class Student extends Person {  
    public Student() {  
        super();  
    }  
}
```

```
public class Test {  
    public static void main(String[] args) {  
        Student s=new Student();  
    }  
}
```

# 상속 실습 A



다음 조건을 만족하는 클래스 Person을 작성

- 필드변수명(설명,자료형): id (학번, String)
- id를 파라미터로 전달받아 대응하는 필드에 저장하는 생성자
- id 값을 반환하는 toString()



다음 조건을 만족하는 클래스 Student를 작성

- 클래스 Person을 상속
- 필드변수명(설명,자료형): name (이름, String)
- id, name 값을 파라미터로 전달받아 대응하는 필드에 저장하는 생성자
  - ◆ Person 클래스의 생성자 호출할 것
- id, name 값을 다음 형식으로 반환하는 toString() 작성 (Person 클래스의 toString() 호출)
  - ◆ 예) 학번: P000123456, 이름: 김영희



클래스 Test의 main 메소드 내 다음 절차 수행 코드 작성

- 다음 표에 대응하는 Student 객체 생성
- 위 Student 객체 정보 출력

id	name
P000123456	김영희

## 상속 실습 B

- 다음 조건을 만족하는 고객 클래스 Customer를 작성
  - 필드변수명(설명,자료형): email(메일주소, String), eMoney(사이버캐쉬,double)
  - email을 파라미터로 전달받아 대응하는 필드에 저장하는 생성자
  - 파라미터로 전달받은 결재금액(정수 int)의 10%를 eMoney에 누적하는 메소드 void pay(int money)
  - 아래 실행 결과 참조하여 toString() 정의
- 다음 조건을 만족하는 VIP고객 클래스 VIPCustomer를 클래스 Customer를 상속받아 작성
  - Email을 파라미터로 전달받아 대응하는 필드에 저장하는 생성자: Customer 클래스의 생성자 호출할 것
  - 파라미터로 전달받은 결재금액(정수 int)의 30%를 eMoney에 누적하는 메소드 void pay(int money)
    - ◆ Customer 클래스의 pay() 메소드를 오버라이드하여 정의할 것
  - 아래 실행 결과 참조하여 toString() 정의
- 다음은 Customer 객체 및 VIPCustomer 객체 생성 후 각각 만원을 결제한 후 각 객체를 출력하는 코드와 그 실행 결과이다. 아래 코드와 실행 결과를 참조하여 Customer 및 VIPCustomer 클래스를 완성하시오.

```
public class Test {  
    public static void main(String[] args) {  
        Customer c=new Customer("sun@ks.ac.kr");  
        VIPCustomer v=new VIPCustomer("mars@ks.ac.kr");  
        c.pay(10000); // 만원 결재  
        v.pay(10000); // 만원 결재  
        System.out.println(c);  
        System.out.println(v);  
    }  
}
```

### 실행결과

Customer: 메일주소=sun@ks.ac.kr, eMoney=1000.0  
VIPCustomer: 메일주소=mars@ks.ac.kr, eMoney=3000.0

# 상속 실습 C



다음 조건을 만족하는 클래스 Book을 작성

- 필드변수명(설명,자료형): id (도서고유번호, String), title (서명, String), year (출판년도, int)
- id, title, year 값을 파라미터로 전달받아 대응하는 필드에 저장하는 생성자 작성
- id, title, year 값을 다음 형식으로 반환하는 toString() 작성
  - ◆ 예) B00012, 자바 시작하기, 2018(년)



다음 조건을 만족하는 클래스 EBook을 작성

- 클래스 Book을 상속
- 필드변수명(설명,자료형): fileSize (파일크기, double)
- id, title, year, fileSize를 파라미터로 전달받아 대응하는 필드에 저장하는 생성자 작성
  - ◆ Book 클래스의 생성자 호출할 것
- id, title, year, fileSize 값을 다음 형식으로 반환하는 toString() 작성. Book 클래스의 toString() 호출할 것
  - ◆ 예) B00012, 자바 시작하기, 2018(년), 20.4(MB)



클래스 Test의 main 메소드 내 다음 절차 수행 코드 작성

- 다음 표에 대응하는 EBook 객체 생성

id	title	year	fileSize (MB)
B00012	자바 시작하기	2018	20.4

- 위 EBook 객체 정보 출력

## 상속 실습 D

- 다음 조건에 따라 직원 클래스 Employee를 작성
  - 필드변수명(설명,자료형): id (직원고유번호, String), monthlyPay (기본 월 급여액, double, 초기값 100(만원))
  - id 값을 파라미터로 전달받아 대응하는 필드에 저장하는 생성자 작성
  - 메소드 getMonthlyPay() → 필드 변수 monthlyPay 값을 반환
- Employee를 상속 받아 내근 직원 클래스 IndoorEmployee를 아래와 같이 작성하시오
  - 필드변수명(설명,자료형): certificateLevel (자격등급, int)
  - id, certificateLevel 값을 파라미터로 전달받아 대응하는 필드에 저장하는 생성자 작성 (부모클래스 생성자 호출)
  - 월급여액 반환하는 getMonthlyPay() 작성 (부모클래스 getMonthlyPay() 오버라이드 할 것)
    - ◆ 내근 직원 월급여액 = 기본월급여액 + 기본월급여액\*자격등급/100
- Employee를 상속 받아 외근 직원 클래스 OutdoorEmployee를 아래와 같이 작성하시오
  - 필드변수명(설명,자료형): dangerLevel (위험등급, int)
  - id, dangerLevel 값을 파라미터로 전달받아 대응하는 필드에 저장하는 생성자 작성 (부모클래스 생성자 호출)
  - 월급여액 반환하는 getMonthlyPay() 작성 (부모클래스 getMonthlyPay() 오버라이드 할 것)
    - ◆ 외근 직원 월급여액 = 기본월급여액 + 기본월급여액\*위험등급/10
- 클래스 Test의 main 메소드 내 다음 절차 수행 코드 작성
  - 다음 표에 대응하는 직원 정보를 각각 IndoorEmployee, OutdoorEmployee 객체 변수 e1, e2에 저장
  - e1, e2의 월급여 출력

id	certificateLevel	비고
E-I-222	2	내근 직원
E-O-777	4	외근 직원



# 상속 실습 E

- ✚ 다음 조건을 만족하는 은행계좌 클래스 BankAccount를 작성하시오
  - 필드변수명(설명,자료형): id (계좌번호, String), balance (잔고, double)
  - id, balance 값을 파라미터로 전달받아 대응하는 필드에 저장하는 생성자 작성
  - 입금액 money를 파라미터로 전달받아 balance에 누적하는 메소드 void deposit(double money) 작성
  - 출금액 money를 파라미터로 전달받아 balance에서 차감하는 메소드 void withdraw(double money) 작성
  - id, balance를 반환하는 toString() 작성
- ✚ 다음 조건을 만족하는 포인트 지원 은행계좌 클래스 PointBankAccount를 작성하시오
  - 클래스 BankAccount를 상속
  - 필드변수명(설명,자료형): point (포인트, double)
  - id, balance, point 값을 파라미터로 전달받아 대응하는 필드에 저장하는 생성자 작성
    - ◆ BankAccount 클래스의 생성자 호출할 것
  - id, balance, point 값을 반환하는 toString() 작성
    - ◆ BankAccount 클래스의 toString() 호출할 것
  - BankAccount 클래스의 void deposit(double money)를 오버라이드하는 메소드 작성
    - ◆ 입금액 money를 balance에 누적하는 것은 동일하며 추가적으로 입금액의 10%를 point에 누적
    - ◆ (추가) 입금액 money를 balance에 누적하기 위해 BankAccount 클래스의 deposit()을 호출하는 방식으로 구현해 보시오
- ✚ 클래스 Test의 main 메소드 내 다음 절차 수행 코드 작성
  - 계좌번호가 KSB-0123이고 잔고 0원, 포인트 0원의 PointBankAccount 객체를 변수 pba에 저장
  - 위 pba 은행계좌객체에 35000원 입금(deposit), 150000원 입금(deposit), 58000원 출금(withdraw) 수행
  - pba 객체의 잔고 출력

# 상속 관계 객체 참조

```
public class Person {  
    String name;  
    public Person(String name) {  
        this.name=name;  
    }  
}
```



```
public class Person extends Object {  
    String name;  
    public Person(String name) {  
        this.name=name;  
    }  
}
```

Object  
|  
Person

Person is a Object (O)  
Object is a Person (X)

```
public class Test {  
    public static void main(String[] args) {  
  
        Person person=new Object();  
  
        Object object=new Person("이영희");  
  
    }  
}
```

← 자식클래스 참조변수로  
부모클래스 객체 참조 불가

← 부모클래스 참조변수로  
자식클래스 객체 참조 OK

# 상속 관계 객체 참조

```
public class Person {  
    String name;  
    public Person(String name) {  
        this.name=name;  
    }  
    @Override  
    public String toString() {  
        return "사람";  
    }  
}
```

```
public class Student extends Person {  
    int score;  
    public Student(String name, int score) {  
        super(name);  
        this.score=score;  
    }  
    @Override  
    public String toString() {  
        return "학생";  
    }  
}
```

```
public class Teacher extends Person {  
    String subject;  
    public Teacher(String name, String subject) {  
        super(name);  
        this.subject=subject;  
    }  
    @Override  
    public String toString() {  
        return "선생님";  
    }  
}
```

Person  
|  
Student

```
Student s=new Person("이영희");
```

← 자식클래스 참조변수로  
부모클래스 객체 참조?

*Student is-a Person (O)*  
*Person is-a Student (X)*

```
Person p=new Student("홍길동", 98);
```

← 부모클래스 참조변수로  
자식클래스 객체 참조?

# 상속 관계 객체 참조

```
public class Person {  
    String name;  
    public Person(String name) {  
        this.name=name;  
    }  
    @Override  
    public String toString() {  
        return name;  
    }  
}
```

```
public class Student extends Person {  
    int score;  
    public Student(String name, int score) {  
        super(name);  
        this.score=score;  
    }  
    @Override  
    public String toString() {  
        return "학생: "+super.toString()+" "+score;  
    }  
}
```

```
public class Teacher extends Person {  
    String subject;  
    public Teacher(String name, String subject) {  
        super(name);  
        this.subject=subject;  
    }  
    @Override  
    public String toString() {  
        return "선생님: "+super.toString()+" "+subject;  
    }  
}
```

```
public class Test {  
    public static void main(String[] args) {  
        Student s=new Student("김철수", 90);  
        Teacher t=new Teacher("이영희", "수학");  
        Person persons[]={s, t};  
        Object objects[]={s, t};  
        println(persons);  
        println(objects);  
    }  
    private static void println(Object[] objects) {  
        for (Object o : objects) System.out.println(o);  
    }  
    private static void println(Person[] persons) {  
        for (Person p : persons) System.out.println(p);  
    }  
}
```

## 실행결과

학생: 김철수,90  
선생님: 이영희,수학  
학생: 김철수,90  
선생님: 이영희,수학

# 상속 관계 객체 참조: type casting, instanceof

```
public class Person {
    String name;
    public Person(String name) {
        this.name=name;
    }
    @Override
    public String toString() {
        return name;
    }
}
```

```
public class Student extends Person {
    int score;
    public Student(String name, int score) {
        super(name);
        this.score=score;
    }
    @Override
    public String toString() {
        return "학생: "+super.toString()+" "+score;
    }
}
```

```
public class Teacher extends Person {
    String subject;
    public Teacher(String name, String subject) {
        super(name);
        this.subject=subject;
    }
    @Override
    public String toString() {
        return "선생님: "+super.toString()+" "+subject;
    }
}
```

```
public class Test {
    public static void main(String[] args) {
        // 자식클래스 참조변수는 부모클래스 객체 참조 불가
        // Student s=new Person("이영희"); // 오류

        // 부모클래스 참조변수는 자식클래스 객체 참조 가능
        Person p=new Student("홍길동", 98);
        System.out.println(p); // 학생 (컴파일 시 p는 Person, 실행 시 Student)
        System.out.println(p.name); // 홍길동
        // System.out.println(p.score); // 컴파일 오류, score 필드 접근하려면?

        Student s=(Student) p; // p의 참조 객체가 Student인 것을 알고 있다면
        System.out.println(s.score);
        // Teacher t=(Teacher) p; // 실행 시 ClassCastException 발생
        // System.out.println(t.subject);

        printInfo(p); // p는 Student 객체를 참조하고 있음
        p=new Teacher("이영희", "수학");
        printInfo(p); // p는 Teacher 객체를 참조하고 있음
    }
}
```

```
private static void printInfo(Person p) {
    // p의 참조 객체가 Student인지 Teacher인지 불명확 (확인 필요)
    if(p instanceof Student){
        Student s=(Student)p;
        System.out.println("학생 성적:"+s.score);
    }
    if(p instanceof Teacher){
        Teacher t=(Teacher)p;
        System.out.println("선생님 담당 과목:"+t.subject);
    }
}
```

**실습:** 다음 Student 객체와 Teacher 객체를 배열에 저장한 후 반복문을 통해 배열 내 객체를 출력하는 코드를 작성하시오

- Student 객체(name:홍길동, score:95)
- Teacher 객체(name:마이클, subject:영어)

## 상속 실습 F

- ✚ 클래스 Shape(도형)을 다음 조건에 따라 정의하시오
  - 실수 0.0을 반환하는 메소드 `getArea()`
- ✚ 클래스 Rectangle(사각형)을 클래스 Shape을 상속받아 다음 조건에 따라 정의하시오
  - 필드: `width`(가로 길이, 정수), `height`(세로 길이, 정수)
  - `width`, `height` 값을 파라미터로 전달받아 대응하는 필드에 저장하는 생성자
  - 사각형의 면적을 실수로 반환하는 메소드 `getArea()`을 Shape의 `getArea()`를 오버라이드하여 작성
- ✚ 클래스 Circle(원)을 클래스 Shape을 상속받아 다음 조건에 따라 정의하시오
  - 필드: `radius`(반지름, 정수)
  - `radius` 값을 파라미터로 전달받아 대응하는 필드에 저장하는 생성자
  - 원의 면적을 실수로 반환하는 메소드 `getArea()`을 Shape의 `getArea()`를 오버라이드하여 작성
- ✚ 클래스 Test의 `main()` 내에 다음 절차를 코딩하시오
  - 다음 객체들을 **Shape 배열**에 저장
    - ◆ Rectangle 객체(가로 3, 세로 4), Circle 객체(반지름 5), Circle 객체(반지름 2)
  - 반복문을 통해 배열에 저장된 각 도형의 면적(`getArea()` 호출)을 출력
  - 반복문을 통해 배열에 저장된 각 도형의 면적(`getArea()` 호출)을 도형 유형(사각형, 원)과 함께 출력


## 상속 실습 G

- ✚ 클래스 Rectangle(사각형)을 다음 조건에 따라 정의하시오
  - 필드: width(가로 길이, 정수), height(세로 길이, 정수)
  - width, height 값을 파라미터로 전달받아 대응하는 필드에 저장하는 생성자
  - 사각형의 면적을 실수로 반환하는 메소드 getArea()
- ✚ 클래스 Circle(원)을 다음 조건에 따라 정의하시오
  - 필드: radius(반지름, 정수)
  - radius 값을 파라미터로 전달받아 대응하는 필드에 저장하는 생성자
  - 원의 면적을 실수로 반환하는 메소드 getArea()
- ✚ 클래스 Test의 main() 내에 다음 절차를 코딩하시오
  - 다음 객체들을 배열에 저장
    - ◆ Rectangle 객체(가로 3, 세로 4), Circle 객체(반지름 5), Circle 객체(반지름 2)
  - 반복문을 통해 배열에 저장된 각 도형의 면적(getArea() 호출)을 도형 유형(사각형, 원)과 함께 출력

## References

 <http://docs.oracle.com/javase/7/docs/api/>

 <https://docs.oracle.com/javase/tutorial/java/>

 김윤명. (2008). 뇌를 자극하는 Java 프로그래밍. 한빛미디어.

 남궁성. 자바의 정석. 도우출판.

 황기태, 김효수 (2015). 명품 Java Programming. 생능출판사.