캡슐화와 정보은닉

이찬수
chansu@sarammani.com

캡슐화(Encapsulation)

- 데이터와 그와 관련된 연산(함수)들을 하나의 객체로 묶음
  - 보다 쉽고 안전한 사용 가능

- 반드시 하나의 클래스로 만들어야 하는 건 아님

캡슐화의 이점

A 클래스가 캡슐화가 잘 되어있다면,
A 클래스가 변경되더라도, A와 연관된
나머지 클래스는 변경되지 않거나
변경되더라도 그 범위가 매우 최소화 됨

멤버필드에 대한 부적절한 접근 예

```
1
2  public class CircleTest2 {
3
4      public static void main(String[] args) {
5          Circle c1;
6          c1 = new Circle();
7          c1.radius = 1;
8
9          Circle c10 = new Circle();
10         c10.radius = 10;
11
12         System.out.print("반지름 " + c1.radius + "인 ");
13         System.out.print("원의 둘레는 " + c1.getCircumference());
14
15         c1.radius = -99;          // 갑자기 원의 반지름이 음수?
16
17         System.out.println("원의넓이는 " + c1.getArea());
18
19         System.out.print("반지름 " + c10.radius + "인 ");
20         System.out.print("원의 둘레는 " + c10.getCircumference());
21         System.out.println("원의넓이는 " + c10.getArea());
22     }
23
24 }
```

정보은닉(Information Hiding)

- 객체 외부에 인터페이스만 드러내고 객체의 내부 구현은 숨김
  - 멤버 변수 선언시 private 접근제어 지정자 이용
  - 메서드를 통한 안전한 접근만 허용

```
1
2  public class Circle {
3      static final double PI = 3.14159265;
4      private double radius;
5
6      double getCircumference() {
7          return 2 * PI * radius;
8      }
9
10     double getArea() {
11         return PI * radius * radius;
12     }
13
14     static double getPI() {
15         return PI;
16     }
17
18 }
```

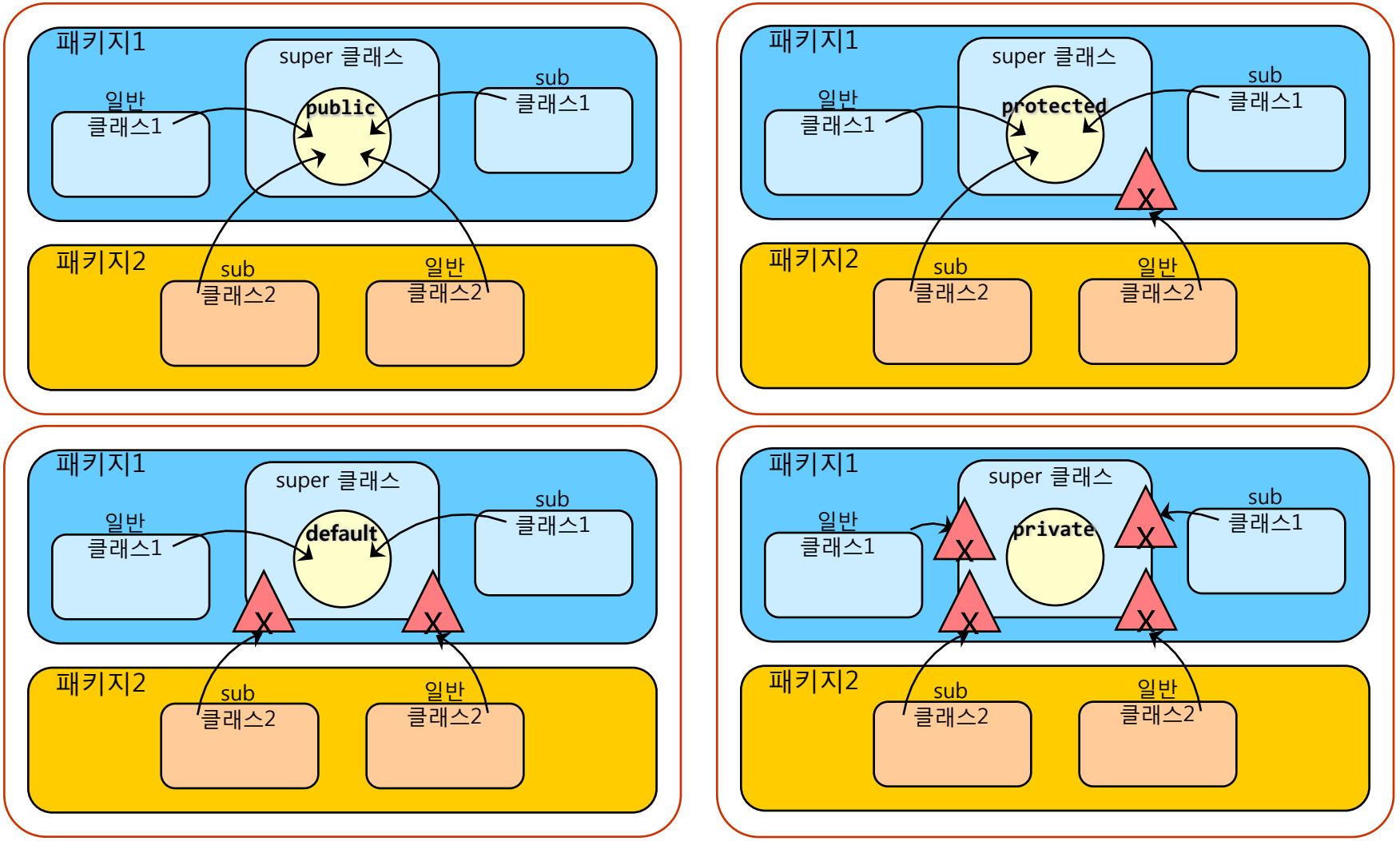
```
1
2  public class CircleTest2 {
3
4      public static void main(String[] args) {
5          Circle c1;
6          c1 = new Circle();
7          c1.radius = 1;
```

각종 변수의 기본 유효 범위

- 지역변수와 매개변수
  - 해당 메서드 안에서만 사용 가능
  - 메서드 시작시 자동 할당되고,
    메서드 반환처리 과정에서 자동 해제됨

- 멤버 변수
  - 모든 멤버 변수들은 그 클래스 객체 전체에서 유효한 범위를 가짐
  - 객체가 생성되면 할당되고, 객체가 소멸되면 해제됨

멤버 접근 제한


패키지1
super 클래스
public
일반
클래스1
sub
클래스1
패키지2
sub
클래스2
일반
클래스2
패키지1
super 클래스
protected
일반
클래스1
sub
클래스1
X
패키지2
sub
클래스2
일반
클래스2
패키지1
super 클래스
default
일반
클래스1
sub
클래스1
X
X
패키지2
sub
클래스2
일반
클래스2
패키지1
super 클래스
private
일반
클래스1
sub
클래스1
X
X
X
X
패키지2
sub
클래스2
일반
클래스2

Access 메서드

- (필요에 따라)특정 멤버 변수의 값을 얻어오거나 설정하는 메서드
  - 멤버 변수를 객체 외부에서 직접 조작하는 대신 getter나 setter를 통한 안전한 접근 제공
    - getter
      - 지정된 멤버 변수의 값을 반환
    - setter
      - 지정된 멤버 변수를 실인수의 값으로 설정
  - 보통 당장 필요하지 않더라도 필요할 것으로 예상되면 멤버에 포함시킴

default 클래스와 public 클래스

- default 클래스
  - 같은 패키지 내부에서만 객체 생성이 가능

- public 클래스
  - 어디서나 클래스 객체 생성을 허용
    - 클래스 정의시 `class` 키워드 앞에 `public` 추가
  - 하나의 소스 파일에는 하나의 클래스만 public 이 될 수 있다
    - 소스 파일의 이름과 반드시 일치해야 함

캡슐화와 정보 은닉

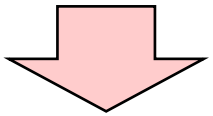
- 데이터와 그와 관련된 연산들을 하나의 객체로 묶음
  - 보다 쉽고 안전한 사용 가능
  - 정보은닉이 포함된 개념

- 일련의 복잡한 행위들을 간단히 수행시킬 수 있는 public 메서드 제공
  - 패키지 내의 모든 클래스를 public 클래스로 정의할 필요는 없음
  - 클래스 내의 모든 멤버들을 public 멤버로 정의할 필요는 없음

생성자

원 클래스의 개선

- 2개의 원 객체
  - 각 원의 반지름이 서로 다르다
    - c1: 반지름 1
    - c10: 반지름 10



- 각 원 객체의 instance 변수의 초기값이 달라야 함
  - 각 원의 반지름을 설정하는 '안전한' 방법 제공 필요

수정된 원 클래스

- 멤버변수 접근을 위한 액세스 메서드 추가

```
1
2  public class Circle {
3      static final double PI = 3.14159265;
4      private double radius;
5
6      double getCircumference() {
7          return 2 * PI * radius;
8      }
9
10     double getArea() {
11         return PI * radius * radius;
12     }
13
14     static double getPI() {
15         return PI;
16     }
17
18     double getRadius() {
19         return radius;
20     }
21
22     void setRadius(double radius) {
23         this.radius = radius;
24     }
25
26 }
```

```
1
2  public class CircleTest2 {
3
4      public static void main(String[] args) {
5          Circle c1;
6          c1 = new Circle();
7          c1.setRadius(1);
8
9          Circle c10 = new Circle();
10         c10.setRadius(10);
11
12         System.out.print("반지름 " + c1.getRadius() + "인 ");
13         System.out.print("원의 둘레는 " + c1.getCircumference());
14
15         c1.setRadius(-1);          // 여전히 부적절한 설정
16
17         System.out.println("원의넓이는 " + c1.getArea());
18
19         System.out.print("반지름 " + c10.getRadius() + "인 ");
20         System.out.print("원의 둘레는 " + c10.getCircumference());
21         System.out.println("원의넓이는 " + c10.getArea());
22     }
23
24 }
```

- 각각의 유효한 객체 생성을 위해 2문장 필요
  → 객체생성과 초기화
- 초기화 메서드가 2번 이상 호출될 수 있음
  → 객체 초기화는 한번만 수행되어야 바람직

생성자(Constructor)

- 객체가 생성될 때 단 한 번 자동으로 호출되는 일종의 메서드
  - 객체의 멤버 변수를 '초기화'
  - 명시적으로 따로 호출하지는 않음

- 생성자 구현 조건
  - 생성자 이름은 클래스 이름과 같아야
  - 반드시 반환 자료형을 지정되지 않아야 됨

```
[접근제한] <생성자명>([매개변수1], ..., [매개변수n]) {
   수행문1;
   수행문2;
   ...
}
```

default 생성자

- 컴파일러에 의해 자동으로 제공
  - 프로그래머가 단 하나도 명시적으로 생성자를 정의하지 않은 경우에만 제공됨
  - 프로그래머가 생성자를 하나라도 정의하면 제공되지 않음
- 인자가 없는 생성자
  - 실제로 초기화를 위해 아무 일도 하지 않음!
- 사용 예
  - `Circle c = new Circle();`

매개변수를 가지는 생성자

- 객체가 가질 instance 멤버 변수의 초기 설정값을 객체 생성시 전달 받음
- 객체 생성 후 초기화 작업이 한 번에 수행
  - 초기화 작업에 대해 프로그래머가 신경쓰지 않아도 됨
  - 초기화 작업은 객체 생성시에만 수행됨
- 사용 예
  - `Circle c = new Circle(1);`
    - `Circle c = new Circle();`
      `c.setRadius(1);`

생성자 오버로딩

- 생성자의 다중정의
  - 매개변수 리스트가 서로 다른 생성자들

- 객체 생성 후 다양한 방법으로 객체 초기화 위해 다수의 생성자 제공
  - `Circle c1 = new Circle(1);`
  - `Circle c2 = new Circle();`

this([*인수리스트*]);

- 반드시 기존 생성자의 첫 행에 위치해야
- 현재 클래스의 인수 리스트가 일치하는 생성자의 호출을 의미
- 한 클래스 내의 특정 생성자 구현시 오버로딩되어 있는 다른 생성자를 호출할 때 사용

완성된 원 클래스

```
1
2  public class Circle {
3      public static final double PI = 3.14159265;
4      private double radius;
5
6      public double getCircumference() {
7          return 2 * PI * radius;
8      }
9
10     public double getArea() {
11         return PI * radius * radius;
12     }
13
14     /*
15     static double getPI() {
16         return PI;
17     }
18     */
19
20     public double getRadius() {
21         return radius;
22     }
23
24     /*
25     void setRadius(double radius) {
26         this.radius = radius;
27     }
28     */
```

```
30     public Circle(double radius) {
31         this.radius = radius;
32     }
33
34     public Circle() {
35         this(1);
36     }
37 }
```

원 객체의 안전한 초기화

```
1
2  public class CircleTest2 {
3
4      public static void main(String[] args) {
5          Circle c1;
6          c1 = new Circle();
7
8          Circle c10 = new Circle(10);
9
10         System.out.print("반지름 " + c1.getRadius() + "인 ");
11         System.out.print("원의 둘레는 " + c1.getCircumference());
12
13 //      c1.setRadius(-1);          // 여전히 부적절한 설정 차단
14
15         System.out.println("원의넓이는 " + c1.getArea());
16
17         System.out.print("반지름 " + c10.getRadius() + "인 ");
18         System.out.print("원의 둘레는 " + c10.getCircumference());
19         System.out.println("원의넓이는 " + c10.getArea());
20     }
21
22 }
```

가비지 컬렉션

가비지 컬렉션

- JAVA는 소멸자를 가지고 있지 않다!
  - C++의 소멸자
    - new에 의해 할당된 메모리를 다시 반납하는 과정을 처리하는 메서드
- 대신 JVM에 가비지 컬렉터(garbage collector)를 포함
  - 필요 없는 객체를 찾아 자동으로 제거
    - 보통 참조 개수가 0인 영역을 제거

내부 클래스

내부 클래스(Inner class)

- 한 클래스 정의 안에 정의된 또다른 클래스
  - 내부 클래스의 메서드에서 외부 클래스의 멤버에 직접 접근이 가능
  - 선언된 위치에 따라 변수와 동일한 유효 범위와 접근성을 가짐
  - 코드의 복잡성을 줄일 수 있음

내부 클래스의 예

```
1
2  public class Outer {
3      String name = "outer";
4      Inner inner = new Inner();
5
6      void ShowOuterName() {
7          System.out.println(name);
8          inner.ShowInnerName();
9      }
10
11
12     class Inner {
13
14         String iname = "inner";
15
16         void ShowInnerName() {
17             System.out.println(name + "-" + iname);
18         }
19     }
20
21
22     public static void main(String[] args) {
23         Outer o = new Outer();
24         o.ShowOuterName();
25     }
26 }
```

수고했습니다!

- 교재 관련 단원
  - 4.7 접근지정자
  - 4.3 생성자
  - 4.6 객체의 소멸