

추상클래스와 인터페이스

추상 클래스

- 추상 메소드(abstract method)
 - abstract로 선언된 메소드, 메소드의 코드는 없고 원형만 선언

```
public abstract String getName(); // 추상 메소드
```



```
public abstract String fail() { return "Good Bye"; } // 추상 메소드 아님. 컴파일 오류
```

- 추상 클래스(abstract class)
 - 추상 메소드를 가지며, abstract로 선언된 클래스
 - 추상 메소드 없이, abstract로 선언한 클래스

```
// 추상 메소드를 가진 추상 클래스
abstract class Shape {
    Shape() { ... }
    void edit() { ... }

    abstract public void draw(); // 추상 메소드
}
```


```
// 추상 메소드 없는 추상 클래스
abstract class JComponent {
    String name;
    void load(String name ) {
        this.name= name;
    }
}
```



```
class fault { // 오류. 추상 메소드를 가지고 있으므로 abstract로 선언되어야 함
    abstract void f(); // 추상 메소드
}
```

추상 클래스의 인스턴스 생성 불가

- 추상 클래스는 온전한 클래스가 아니기 때문에 **인스턴스를 생성할 수 없음**

JComponent p;	// 오류 없음. 추상 클래스의 레퍼런스 선언
p = new JComponent();	// 컴파일 오류. 추상 클래스의 인스턴스 생성 불가
 Shape obj = new Shape();	// 컴파일 오류. 추상 클래스의 인스턴스 생성 불가

컴파일 오류 메시지

Unresolved compilation problem: Cannot instantiate the type Shape

추상 클래스의 상속과 구현

- 추상 클래스 상속

- 추상 클래스를 상속받으면 추상 클래스가 됨
- 서브 클래스도 abstract로 선언해야 함

```
abstract class A { // 추상 클래스
    abstract int add(int x, int y); // 추상 메소드
}
abstract class B extends A { // 추상 클래스
    void show() { System.out.println("B"); }
}
```

오류

```
A a = new A(); // 컴파일 오류. 추상 클래스의 인스턴스 생성 불가
B b = new B(); // 컴파일 오류. 추상 클래스의 인스턴스 생성 불가
```

- 추상 클래스 구현

- 서브 클래스에서 슈퍼 클래스의 추상 메소드 구현(오버라이딩)
- 추상 클래스를 구현한 서브 클래스는 추상 클래스 아님

```
class C extends A { // 추상 클래스 구현. C는 정상 클래스
    int add(int x, int y) { return x+y; } // 추상 메소드 구현. 오버라이딩
    void show() { System.out.println("C"); }
}
...
C c = new C(); // 정상
```

추상 클래스의 목적

- 추상 클래스의 목적
 - 상속을 위한 슈퍼 클래스로 활용하는 것
 - 서브 클래스에서 추상 메소드 구현
 - 다형성 실현

```
class Shape {  
    public void draw() {  
        System.out.println("Shape");  
    }  
}
```

```
abstract class Shape {  
    public abstract void draw();  
}
```

추상 클래스로 작성

추상 클래스를 상속받아
추상 메소드 draw() 구현

```
class Line extends DObject {  
    public void draw() {  
        System.out.println("Line");  
    }  
}
```

```
class Rect extends DObject {  
    public void draw() {  
        System.out.println("Rect");  
    }  
}
```

```
class Circle extends DObject {  
    public void draw() {  
        System.out.println("Circle");  
    }  
}
```

추상 클래스의 구현 예제

추상 클래스 Calculator를 상속받는 GoodCalc 클래스를 구현하라.

```
abstract class Calculator {  
    public abstract int add(int a, int b);  
    public abstract int subtract(int a, int b);  
    public abstract double average(int[] a);  
}
```

추상 클래스의 구현 예제

```
public class GoodCalc extends Calculator {  
    public int add(int a, int b) { // 추상 메소드 구현  
        return a + b;  
    }  
    public int subtract(int a, int b) { // 추상 메소드 구현  
        return a - b;  
    }  
    public double average(int[] a) { // 추상 메소드 구현  
        double sum = 0;  
        for (int i = 0; i < a.length; i++)  
            sum += a[i];  
        return sum/a.length;  
    }  
  
    public static void main(String [] args) {  
        GoodCalc c = new GoodCalc();  
        System.out.println(c.add(2,3));  
        System.out.println(c.subtract(2,3));  
        System.out.println(c.average(new int [] { 2,3,4 }));  
    }  
}
```

5
-1
3.0

인터페이스의 필요성



A사 제품



B사 제품



C사 제품



D사 제품

정해진 규격(인터페이스)에 맞기
만 하면 연결 가능.
각 회사마다 구현 방법은 다름

정해진 규격(인터페이스)에 맞지
않으면 연결 불가

자바의 인터페이스

- 인터페이스

- 일종의 추상 클래스
- 다른 클래스를 작성하는데 도움 줄 목적으로 작성됨

```
interface 인터페이스이름 {  
    public static final 타입 상수이름=값;  
    public abstract 메서드이름(매개변수 목록);  
    public 타입 메소드명(매개변수 목록){};  
    public static 타입 메서드명(매개변수 목록){};  
}
```

Java8 부터 가능

- **public static final** 멤버 변수는 이를 생략할 수 있음
- **public abstract** 메서드는 이를 생략할 수 있음

인터페이스 상속

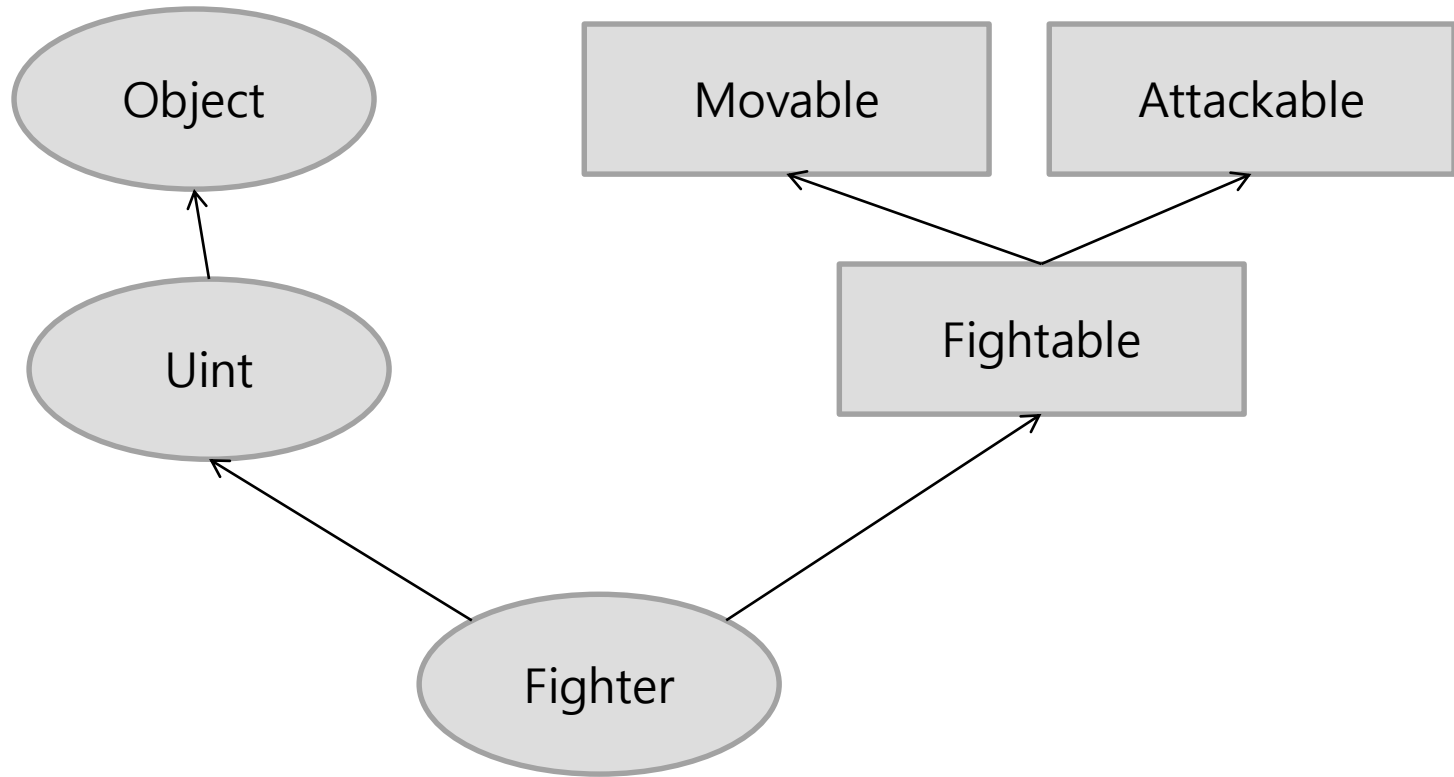
- 인터페이스의 상속
 - 인터페이스로부터만 상속받을 수 있음
 - 클래스와 달리 다중상속이 가능
 - 인터페이스는 클래스와 달리 Object클래스와 같은 최고 조상은 없음

```
interface Movable {  
    void move(int x, int y);  
}
```

```
interface Attackable {  
    void attack(Uint u);  
}
```

```
interface Fightable extends Movable, Attackable { }
```

인터페이스 구현



인터페이스 구현

- 인터페이스의 구현
 - 인터페이스도 추상클래스처럼 인스턴스를 생성할 수 없음
 - 정의된 추상메서드의 몸통을 만들어 주는 클래스 작성해야 함
 - 클래스의 상속과 다르지 않음

```
class 클래스이름 implements 인터페이스이름 {  
  
    //인터페이스에 정의된 추상메서드를 구현해야 함  
  
}
```

```
class Fighter implements Fightable {  
    public void move(){ /*내용 생략*/ }  
    public void attack(){ /*내용 생략*/ }  
}
```

인터페이스 구현

```
abstract class Fighter implements Fightable {  
    public void move(){ /*내용 생략*/ }  
}
```

- 인터페이스 메서드 중 **일부만 구현한다면 추상클래스로** 선언되어야 함
- 클래스 상속과 인터페이스 구현 동시에 수행

```
class Fighter extends Uint implements Fightable {  
    public void move(){ /*내용 생략*/ }  
    public void attack(){ /*내용 생략*/ }  
}
```

인터페이스 구현과 동시에 슈퍼 클래스 상속

```
interface PhoneInterface {
    int BUTTONS = 20;
    void sendCall();
    void receiveCall();
}

interface MobilePhoneInterface
    extends PhoneInterface {
    void sendSMS();
    void receiveSMS();
}

interface MP3Interface {
    public void play();
    public void stop();
}

class PDA {
    public int calculate(int x, int y) {
        return x + y;
    }
}
```

```
// SmartPhone 클래스는 PDA를 상속받고,
// MobilePhoneInterface와 MP3Interface 인터페이스에 선언된
// 메소드를 모두 구현
```

```
class SmartPhone extends PDA implements
    MobilePhoneInterface, MP3Interface {
    public void sendCall() { System.out.println("전화 걸기"); }
    public void receiveCall() { System.out.println("전화 받기"); }
    public void sendSMS() { System.out.println("SMS 보내기"); }
    public void receiveSMS() { System.out.println("SMS 받기"); }

    public void play() { System.out.println("음악 재생"); }
    public void stop() { System.out.println("재생 중지"); }

    public void schedule() { System.out.println("일정 관리"); }
}

public class InterfaceEx {
    public static void main(String [] args) {
        SmartPhone p = new SmartPhone();
        p.sendCall();
        p.play();
        System.out.println(p.calculate(3,5));
        p.schedule();
    }
}
```

MobilePhoneInterface
모든 메소드 구현

MP3Interface의
모든 메소드 구현

새로운
메소드 추가

전화 걸기
음악 재생
8
일정 관리

인터페이스를 이용한 다형성

- 인터페이스 또한 구현한 클래스의 조상

```
Fightable f = (Fightable)new Fighter();
```

```
Fightable f = new Fighter();
```

```
void attack(Fightable f) {  
  
}
```

```
Fightable method() {  
    return new Fighter();  
}
```

인터페이스를 이용한 다중상속

```
public class TV {  
    protected boolean power;  
    protected int channel;  
    protected int volume;  
  
    public void power() { }  
    public void channelUp() { }  
    ...  
}
```

```
public class VCR {  
    protected int counter;  
    public void play() { }  
    public void stop() { }  
    public void reset() { }  
    ....  
}
```

class TVCR

```
graph BT; TVCR --> TV; TVCR --> VCR;
```


인터페이스를 이용한 다중상속

```
public class VCR {  
    protected int counter;  
    public void play() { }  
    public void stop() { }  
    public void reset() { }  
    ....  
}
```



```
public interface IVCR {  
    public void play() { }  
    public void stop() { }  
    public void reset() { }  
    ....  
}
```

```
public class TVCR extends TV implements IVCR
```

인터페이스를 이용한 다중상속

```
public class TVCR extends TV implements IVCR {  
  
    VCR vcr = new VCR();  
  
    public void play(){  
        vcr.play();  
    }  
  
    public void stop() {  
        vcr.stop();  
    }  
  
    .....  
}
```

인터페이스를 이용한 다형성

```
interface Parseable {  
    public abstract void parse(String fileName);  
}  
  
class ParserManager {  
    public static Parseable getParser(String type){  
        if(type.equals("XML")){  
            return new XMLParser();  
        } else {  
            Parseable p = new HTMLParser();  
            return p; // return new HTMLParser();  
        }  
    }  
}
```

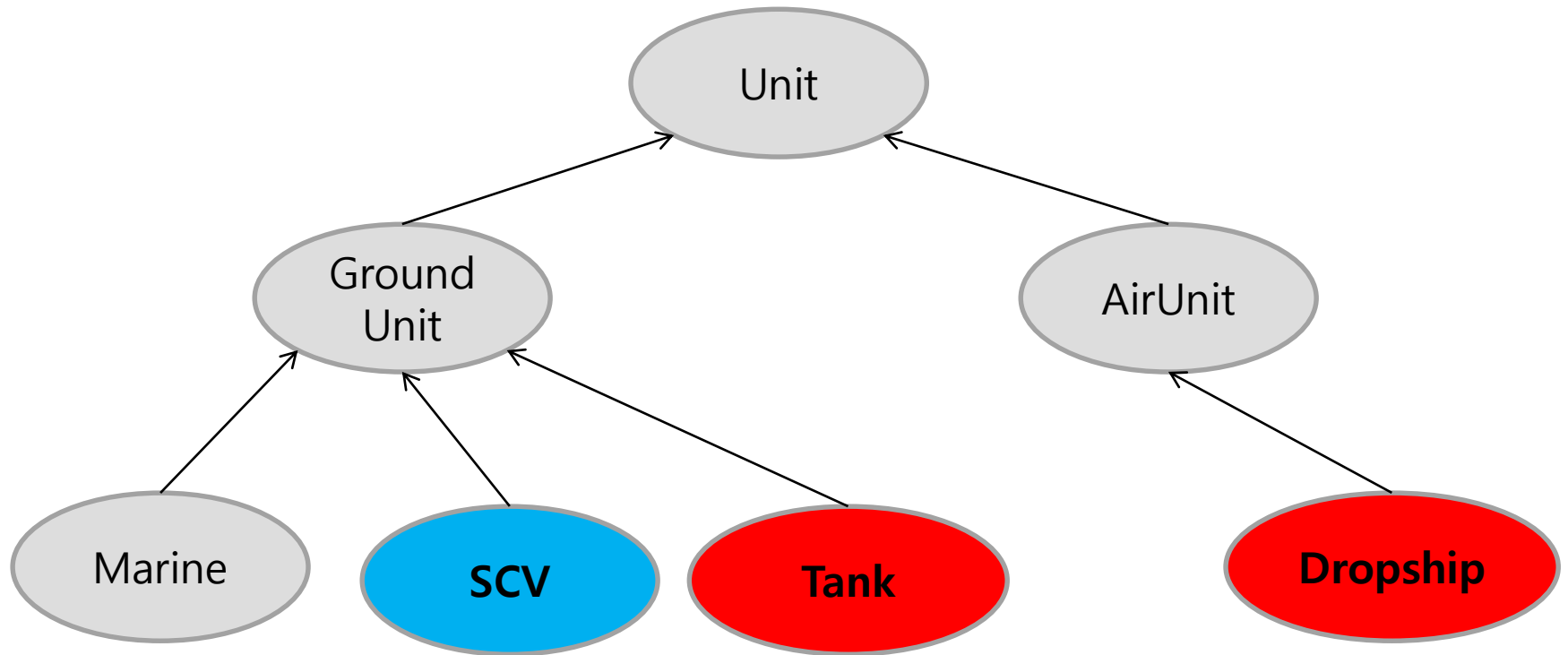
인터페이스를 이용한 다형성

```
class XMLParser implements Parseable {  
    public void parse(String fileName) {  
        System.out.println(fileName+"-XML parsing completed.");  
    }  
}
```

```
class HTMLParser implements Parseable {  
    public void parse(String fileName) {  
        System.out.println(fileName+"-HTML parsing completed.");  
    }  
}
```

```
Parseable parser = ParserManager.getParser("XML");  
parser.parse("document.xml");  
parser = ParserManager.getParser("HTML");  
parser.parse("document.html");
```

인터페이스를 이용한 다형성



인터페이스를 이용한 다형성

```
void repair(Tank t) { }  
void repair(Dropship d) { }
```



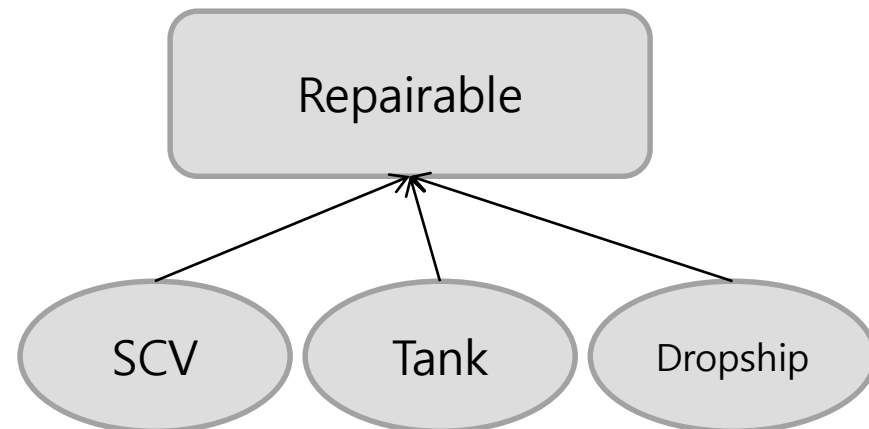
```
void repair(GroundUnit gu) { }  
void repair(AirUnit au) { }
```



```
void repair(Repairable r) { }
```

```
interface Repairable() { }
```

```
class SCV extends GroundUnit  
implements Repairable { }  
class Tank extends GroundUnit  
implements Repairable { }  
class Dropship extends AirUnit  
implements Repairable { }
```



인터페이스의 장점

- 개발시간을 단축시킬 수 있다.
- 표준화가 가능하다.
- 서로 관계없는 클래스들에게 관계를 맺어 줄 수 있다.
- 독립적인 프로그래밍이 가능하다.

중첩 클래스와 인터페이스

중첩 클래스와 중첩 인터페이스

- 중첩 클래스 : 클래스 멤버로 선언된 클래스

```
class ClassName {  
    class NestedClassName {  
    }  
}
```

중첩 클래스

- 중첩 인터페이스 : 클래스 멤버로 선언된 인터페이스
 - UI 컴포넌트 내부 이벤트 처리에 많이 활용

```
class ClassName {  
    interface NestedInterfaceName {  
    }  
}
```

중첩 인터페이스

중첩 클래스

- 중첩 클래스의 분류

선언 위치에 따른 분류		선언 위치	설명
멤버 클래스	인스턴스 멤버 클래스	<pre>class A { class B { ... } }</pre>	A 객체를 생성해야만 사용할 수 있는 B 중첩 클래스
	정적 멤버 클래스	<pre>class A { static class B { ... } }</pre>	A 클래스로 바로 접근할 수 있는 B 중첩 클래스
로컬 클래스		<pre>class A { void method() { class B { ... } } }</pre>	method()가 실행할 때만 사용할 수 있는 B 중첩 클래스

중첩 클래스

- 인스턴스 멤버 클래스

```
class A {  
    /**인스턴스 멤버 클래스**/  
    class B {  
        B() { }                -----생성자  
        int field1;            -----인스턴스 필드  
        //static int field2;    -----정적 필드 (x)  
        void method1() { }     -----인스턴스 메소드  
        //static void method2() { } -----정적 메소드 (x)  
    }  
}
```

```
A    a = new A();  
A.B  b = a.new B();  
b.field1 = 3;  
b.method1();
```

중첩 클래스

- 정적 멤버 클래스
 - static 키워드로 선언된 클래스, 모든 종류의 필드, 메소드 선언 가능

```
class A {  
    /**정적 멤버 클래스**/  
    static class C {  
        C() {}                -----생성자  
        int field1;            -----인스턴스 필드  
        static int field2;      -----정적 필드  
        void method1() {}      -----인스턴스 메소드  
        static void method2() {} -----정적 메소드  
    }  
}
```

```
A.C c = new A.C();  
c.field1 = 3;      //인스턴스 필드 사용  
c.method1();       //인스턴스 메소드 호출  
A.C.field2 = 3;    //정적 필드 사용  
A.C.method2();     //정적 메소드 호출
```

중첩 클래스

- 로컬 클래스 – 메소드 내에서만 사용

```
void method() {  
    /**로컬 클래스**/  
    class D {  
        D() { }                -----생성자  
        int field1;            -----인스턴스 필드  
        //static int field2;    -----정적 필드(x)  
        void method1() { }     -----인스턴스 메소드  
        //static void method2() { } -----정적 메소드(x)  
    }  
    D d = new D();  
    d.field1 = 3;  
    d.method1();  
}
```

```
void method() {  
    class DownloadThread extends Thread { ... }  
    DownloadThread thread = new DownloadThread();  
    thread.start();  
}
```

중첩 클래스

- 바깥 필드와 메서드에서 사용 제한

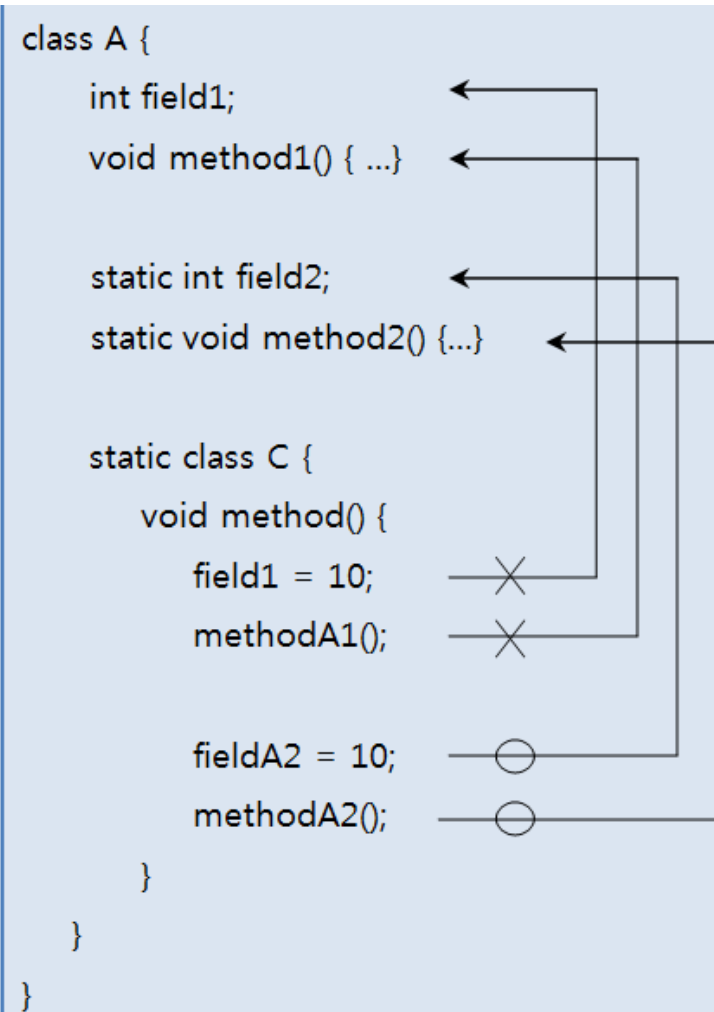
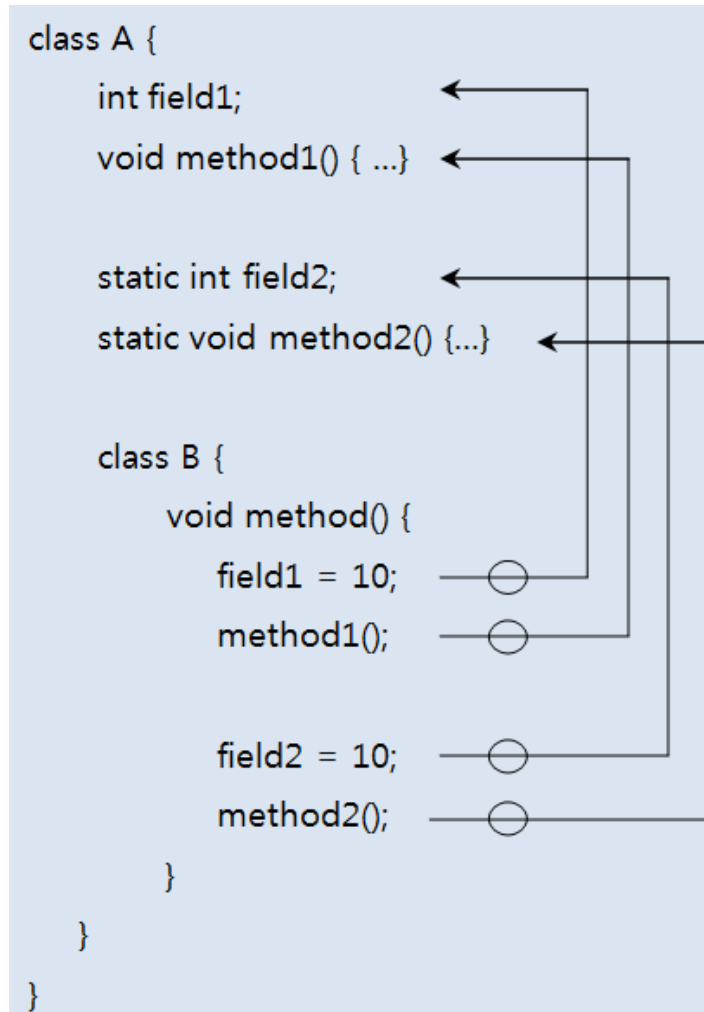
```
public class A {  
    //인스턴스 멤버 클래스  
    class B {}  
  
    //정적 멤버 클래스  
    static class C {}  
}
```

```
public class A {  
    //인스턴스 필드  
    B field1 = new B();           ----- (o)  
    C field2 = new C();           ----- (o)  
  
    //인스턴스 메소드  
    void method1() {  
        B var1 = new B();         ----- (o)  
        C var2 = new C();         ----- (o)  
    }
```

```
    //정적 필드 초기화  
    //static B field3 = new B();   ----- (x)  
    static C field4 = new C();     ----- (o)  
  
    //정적 메소드  
    static void method2() {  
        //B var1 = new B();       ----- (x)  
        C var2 = new C();         ----- (o)  
    }
```

중첩 클래스

- 멤버 클래스에서 사용 제한



중첩 클래스

- 중첩 클래스에서 바깥 클래스 참조

```
public class Outer {
    String field = "Outer-field";
    void method() {
        System.out.println("Outer-method");
    }

    class Nested {
        String field = "Nested-field";
        void method() {
            System.out.println("Nested-method");
        }

        void print() {
            System.out.println(this.field);
            this.method();
            System.out.println(Outer.this.field);
            Outer.this.method();
        }
    }
}
```

중첩 객체 참조

바깥 객체 참조

중첩 클래스

- 로컬 클래스에서 사용 제한

```
public class Outer {  
    //자바7 이전  
    public void method1(final int arg) {  
        final int localVariable = 1;  
        //arg = 100; (x)  
        //localVariable = 100; (x)  
        class Inner {  
            public void method() {  
                int result = arg + localVariable;  
            }  
        }  
    }  
}
```

final 매개변수와 로컬 변수는
로컬 클래스의 메소드의 로컬변수로 복사
(final 붙이지 않으면 컴파일 오류 발생)

```
//자바8 이후  
public void method2(int arg) {  
    int localVariable = 1;  
    //arg = 100; (x)  
    //localVariable = 100; (x)  
    class Inner {  
        public void method() {  
            int result = arg + localVariable;  
        }  
    }  
}
```

중첩 인터페이스

- 중첩 인터페이스 선언

```
public class Button {  
    OnClickListener listener; 인터페이스 타입 필드  
  
    void setOnClickListener(OnClickListener listener) { 매개변수의 다형성  
        this.listener = listener;  
    }  
  
    void touch() {  
        listener.onClick(); 구현 객체의 onClick() 메서드 호출  
    }  
  
    interface OnClickListener { 중첩 인터페이스  
        void onClick();  
    }  
}
```