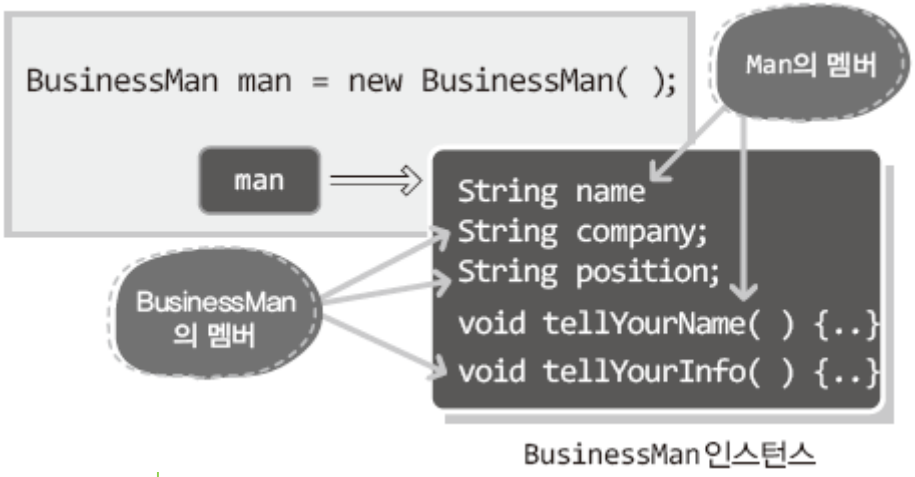
상 속

☞ 상속의 기본 특성

· 기존에 정의된 클래스에 메소드와 변수를 추가하여 새로운 클래스를 정의하는 것!(문법적 측면에서의 상속)

```
class Man
{
    public String name;
    public void tellYourName() {
      System.out.println("My name is "+name); }
}
```

상위클래스, 기초클래스




```
class BusinessMan extends Man
{
    public String company;
    public String position;
    public void tellYourInfo()
    {
        System.out.println("My company is "+company);
        System.out.println("My position is "+position);
        tellYourName();
    }
}
```

Man 클래스를 상속했기 때문에 호출가능!

하위클래스, 유도클래스

☞ 상속 관계에 있는 인스턴스의 생성 예

외부에서 호출하는 것은 BusinessMan 클래스의 생성자이니, 이 생성자가 부모 클래스의 인스턴스 변수를 초기화 할 의무를 지닌다

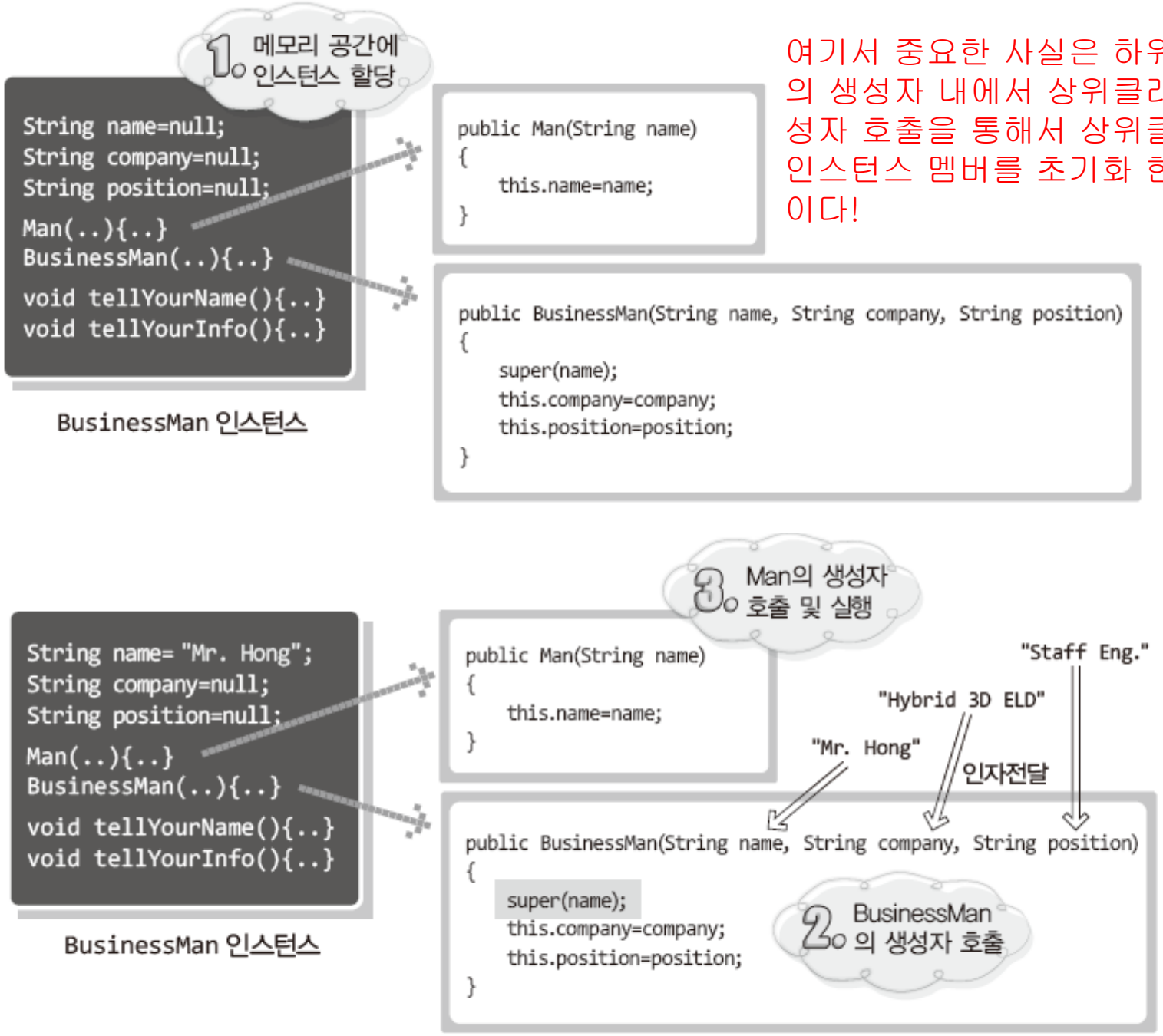
```
class Man {
    String name;
    public Man(String name){this.name=name;}
    public void tellYourName() { System.out.println("My name is "+name); }
}
class BusinessMan extends Man {
    String company;    // 회사이름
    String position;    // 직급
    public BusinessMan(String name, String company, String position) {
        super(name);  // name을 인자로 전달받는 상위클래스의 생성자를 호출하겠다.
        this.company=company;
        this.position=position;
    }
    public void tellYourInfo() {
        System.out.println("My company is "+company);
        System.out.println("My position is "+position);
        tellYourName();  // Man 클래스를 상속했기 때문에 호출 가능!
    }
}
class BasicInheritance {
    public static void main(String[] args) {
        BusinessMan man1 = new BusinessMan("Mr. Hong", "Hybrid 3D ELD", "Staff Eng.");
        BusinessMan man2 = new BusinessMan("Mr. Lee", "Hybrid 3D ELD", "Assist Eng.");
        System.out.println("First man info.....");
        man1.tellYourName();
        man1.tellYourInfo();
        System.out.println("Second man info.....");
        man2.tellYourInfo();
    }
}
```

BusinessMan 인스턴스 생성시 초기화 해야 할 인스턴스 변수는 다음과 같다.

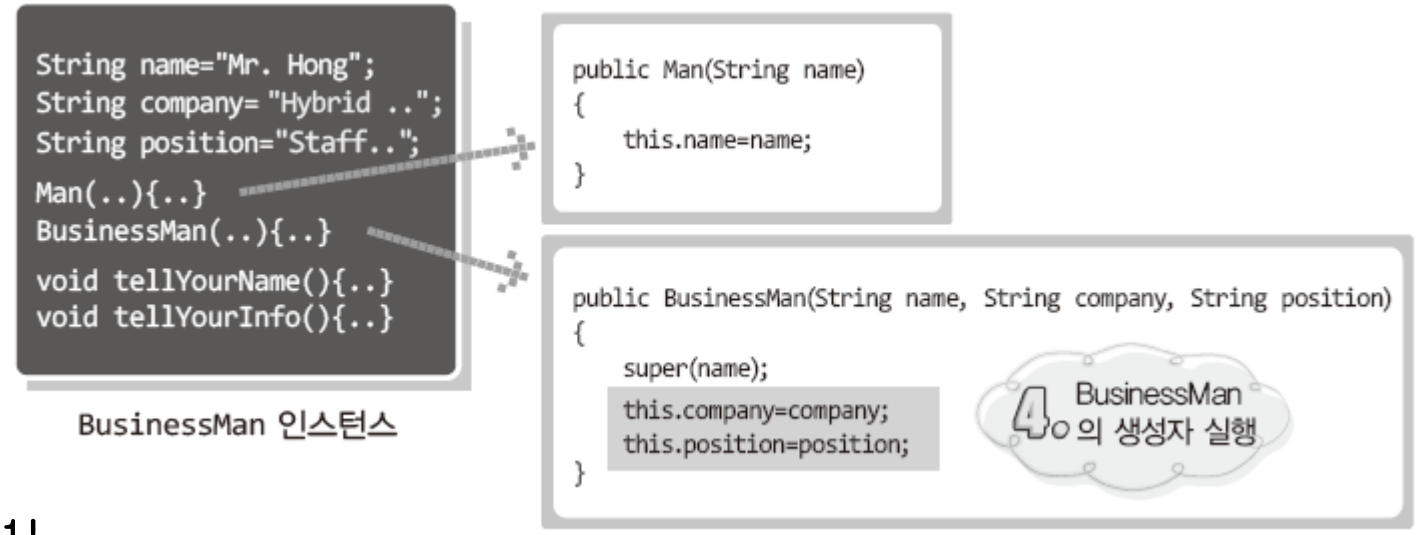
name, company, position

클래스 Man　　클래스 BusinessMan

☞ 상속 관계에 있는 인스턴스의 생성과정(1~3)



☞ 상속 관계에 있는 인스턴스의 생성과정(4)



결론1!

하위 클래스의 생성자는 상위클래스의 인스턴스 변수를 초기화 할 데이터까지 인자로 전달받아야 한다!

결론2!

하위 클래스의 생성자는 상위클래스의 생성자 호출을 통해서 상위클래스의 인스턴스 변수를 초기화 한다!

결론3!

키워드 super는 상위클래스의 생성자 호출에 사용된다. super와 함께 표시된 전달되는 인자의 수와 자료형을 참조하여 호출 할 생성자가 결정된다!

☞ 반드시 호출되어야 하는 상위 클래스의 생성자

```
class AAA
{
    int num1;
}

class BBB extends AAA
{
    int num2;
}
```

AAA() { }

BBB() { super(); }

자동으로 삽입되는 디폴트 생성자의 형태

```
class AAA
{
    int num1;
}

class BBB extends AAA
{
    int num2;
    BBB() { num2=0; }
}
```

super();

자동으로 삽입되는 상위 클래스의 생성자 호출문!

결과적으로 어떠한 형태로건(프로그래머가 직접 삽입하건, 컴파일러가 삽입하건) 상위 클래스의 생성자는 반드시 호출이 이뤄진다!

☞ protected 지시자

지시자	클래스 내부	동일 패키지	상속받은 클래스	이외의 영역
private	●	×	×	×
default	●	●	×	×
protected	●	●	●	×
public	●	●	●	●

- default 패키지로 묶인 두 클래스

```
class AAA
{
    int num1;
    protected int num2;
}

class BBB extends AAA
{
    BBB()
    {
        num1=10;     // AAA 클래스의 default 멤버에 접근
        num2=20;     // BBB 클래스의 protected 멤버에 접근
    }
}
```

이 둘은 상속 관계에 앞서 동일패키지로 묶인 BBB 클래스에 의해 접근 가능!

☞ private 멤버도 상속은 된다. 다만 접근만 불가능!

```
class Accumulator
{
    private int val;

    Accumulator(int init) { val=init; }
    protected void accumulate(int num)
    {
        if(num<0)      // 음수는 누적 대상에서 제외!
            return;
        val += num;
    }
    protected int getAccVal(){return val;}
}
```

private 멤버도 상속이 된다! 다만, 함께 상속된 다른 메소드를 통해서 간접접근을 해야만 한다!

```
class SavingAccount extends Accumulator
{
    public SavingAccount(int initDep)
    {
        super(initDep);      // 통장개설 시 첫 입금액
    }
    public void saveMoney(int money)
    {
        accumulate(money);
    }
    public void showSavedMoney()
    {
        System.out.print("지금까지의 누적금액: ");
        System.out.println(getAccVal());
    }
}
```

☞ static 변수도 상속이 되는가!

•static 변수는 접근의 허용 여부와 관계가 있다. 따라서 다음과 같이 질문을 해야 옳다!

"상위 클래스의 static 변수를 하위 클래스도 그냥 이름만으로 접근이 가능한가요?"

=> YES!

```
class Adder
{
    public static int val=0;
    public void add(int num)
    {
        val += num;
    }
}
class AdderFriend extends Adder
{
    public void friendAdd(int num)
    {
        val += num;
    }
    public void showVal()
    {
        System.out.println(val);
    }
}
```

```
class StaticInheritance {
    public static void main(String[] args) {
        Adder ad = new Adder();
        AdderFriend af = new AdderFriend();
        ad.add(1);
        af.friendAdd(3);
        AdderFriend.val += 5;
        af.showVal();
    }
}
```

상위 클래스의 static 변수에 이름으로 직접 접근이 가능!

☞ 상속을 위한 기본 조건인 IS-A 관계의 성립

- 상속 관계에 있는 두 클래스 사이에는 IS-A 관계가 성립해야 한다.
- IS-A 관계가 성립하지 않는 경우에는 상속의 타당성을 면밀히 검토해야 한다.
- IS-A 이외에 HAS-A 관계도 상속으로 표현 가능하다. 그러나 HAS-A를 대신해서 Composition 관계를 유지하는 것이 보다 적절한 경우가 많다.

- 전화기 → 무선 전화기
- 컴퓨터 → 노트북 컴퓨터

무선 전화기는 전화기를 상속한다!
노트북 컴퓨터는 컴퓨터를 상속한다!

- 무선 전화기는 일종의 전화기입니다.
- 노트북 컴퓨터는 일종의 컴퓨터입니다.

- 무선 전화기 is a 전화기.
- 노트북 컴퓨터 is a 컴퓨터.

☞ IS-A 기반 상속의 예

일반적으로 IS-A 관계가 성립되면, 불필요한 상속관계는 형성될 수 있으나, 잘못된 상속관계가 형성된다고는 이야기하지 않는다.

```
class Computer {
    String owner;
    public Computer(String name){    }
    public void calculate() {       }
}

class NotebookComp extends Computer {    // 노트북 컴퓨터는 컴퓨터이다!
    int battary;
    public NotebookComp(String name, int initChag) {    }
    public void charging() {    }
    public void movingCal() {    }
}

class TabletNotebook extends NotebookComp {  // 태블릿은 노트북 컴퓨터이다!
    String regstPenModel;
    public TabletNotebook(String name, int initChag, String pen) {     }
    public void write(String penInfo) {    }
}
```

☞ HAS-A 관계에 상속을 적용한 경우

```
class Gun {
    int bullet;          // 장전된 총알의 수
    public Gun(int bnum) {bullet=bnum;}
    public void shut() {
        System.out.println("BBANG!");
        bullet--;
    }
}


class Police extends Gun {
    int handcuffs;       // 소유한 수갑의 수
    public Police(int bnum, int bcuff) {
        super(bnum);
        handcuffs=bcuff;
    }
    public void putHandcuff() {
        System.out.println("SNAP!");
        handcuffs--;
    }
}
```

경찰은 총을 소유하고 있다!
경찰 has a 총!

상속은 강한 연결고리를 형성한다. 때문에 총을 소유하지 않는 경찰, 또는 총이 아닌 경찰봉을 소유하는 경찰등 다양한 표현에 한계를 보인다는 단점이 있다!

☞ HAS-A 관계에 복합관계를 적용한 경우

```
class Gun {
    int bullet;  // 장전된 총알의 수
    public Gun(int bnum) {bullet=bnum;}
    public void shut() {
        System.out.println("BBANG!");
        bullet--;
    }
}
```

복합관계는 강한 연결고리를 형성하지 않는다. 따라서 소유하던 대상을 소유하지 않을 수도 있고, 소유의 대상을 늘리는 것도 상속보다 훨씬 간단하다. 때문에 HAS-A 관계는 복합관계로 표현한다.

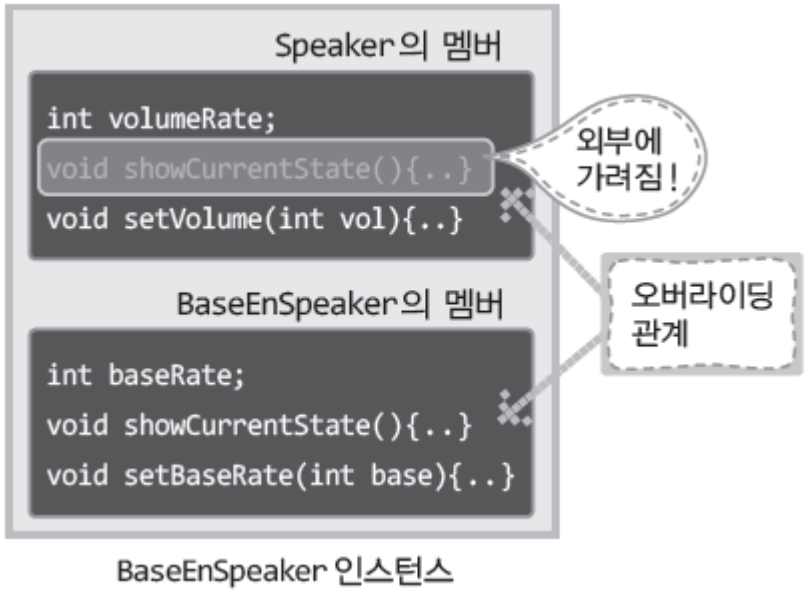
```
class Police
{
    int handcuffs;  // 소유한 수갑의 수
    Gun pistol;  // 소유하고 있는 권총

    public Police(int bnum, int bcuff) {
        handcuffs=bcuff;
        if(bnum!=0)
            pistol = new Gun(bnum);
        else
            pistol=null;
    }
    public void putHandcuff() {
        System.out.println("SNAP!");
        handcuffs--;
    }
    public void shut() {
        if(pistol == null)
            System.out.println("Hut BBANG!");
        else
            pistol.shut();
    }
}
```

☞ 메소드 오버라이딩(Overriding)

- 상위 클래스에 정의된 메소드의 이름, 반환형, 매개 변수 선언까지 완전히 동일한 메소드를 하위 클래스에서 다시 정의하는 것!
- 하위 클래스에 정의된 메소드에 의해 상위클래스의 메소드는 가리워진다.

```
class Speaker {
    private int volumeRate;
    public void showCurrentState() {
        System.out.println("볼륨 크기: "
                                + volumeRate);
    }
    public void setVolume(int vol) {
        volumeRate=vol;
    }
}
```

```
class BaseEnSpeaker extends Speaker
{
    private int baseRate;
    public void showCurrentState() {
        super.showCurrentState();
        System.out.println("베이스 크기: "
                                + baseRate);
    }
    public void setBaseRate(int base) {
        baseRate=base;
    }
```



☞ 상위 클래스의 참조변수로 하위 클래스의 인스턴스 참조

- 중 저음 보강 스피커는 (일종의) 스피커이다. (O)
- BaseEnSpeaker is a Speaker. (O)

- 스피커는 (일종의) 중 저음 보강 스피커이다. (X)
- Speaker is a BaseEnSpeaker. (X)

자바 컴파일러의
실제 관점

```
public static void main(String[] args)
{
    Speaker bs=new BaseEnSpeaker();
    bs.setVolume(10);
    bs.setBaseRate(20);// 컴파일 에러
    bs.showCurrentState();
}
```

BaseEnSpeaker도 Speak의
인스턴스이므로 성립한다!

bs가 참조하는 것은 Speaker의 인스턴스로 인식
하기 때문에 BaseEnSpeaker의 멤버에 접근 불가!

➢ 위의 내용을 정확히 이해하는 것이 중요하다. 특히 상위 클래스의 참조변수가 하위 클래스의 인스턴스를 참조할 수 있는 이유를 잘 이해하자!

☞ 참조변수의 참조 가능성에 대한 일반화

```
class AAA { . . . }
class BBB extends AAA { . . . }
class CCC extends BBB { . . . }
```

아래의 문제제시를 위한 클래스의 상속관계

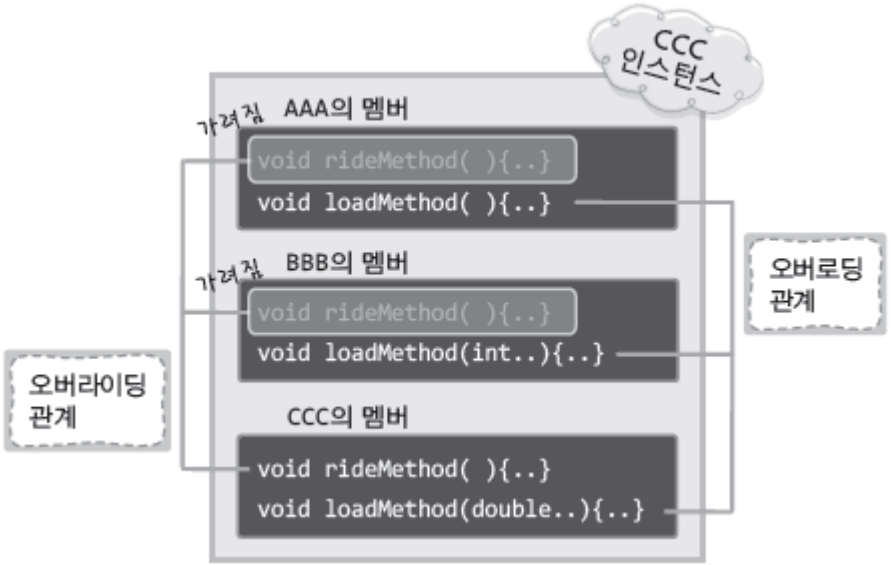
```
AAA ref1 = new BBB();
AAA ref2 = new CCC();
BBB ref3 = new CCC();

CCC ref1 = . . .  // 컴파일 완료
BBB ref2 = ref1;
AAA ref3 = ref1;


AAA ref1 = new CCC();
BBB ref2 = ref1;
CCC ref3 = ref1;
```

- ✓ 참조 변수의 자료형에 따라서 대입연산의 허용여부가 결정된다.
- ✓ 이 사실을 바탕으로 왼쪽 문장 중에서 컴파일 에러가 발생하는 문장들을 모두 고르면?

☞ 오버라이딩 관계에서의 메소드 호출

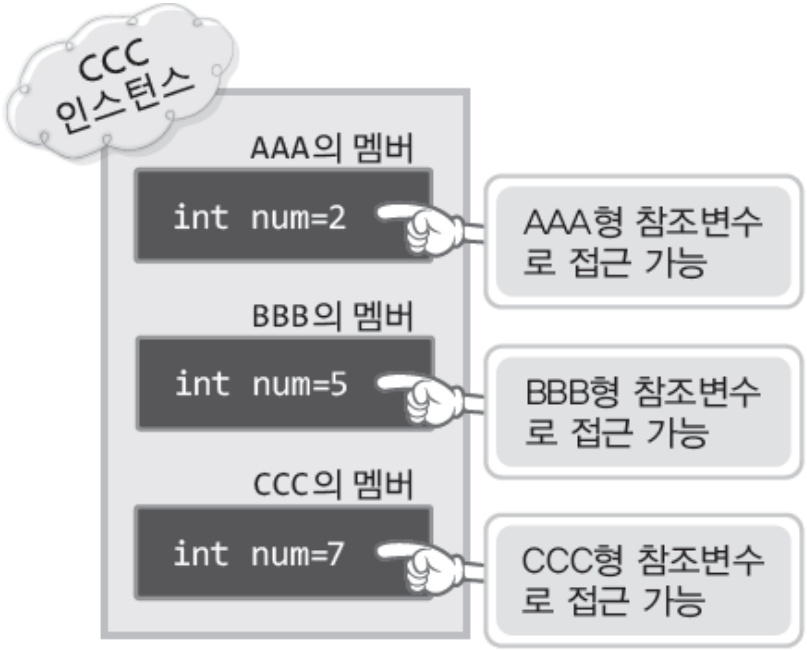
```
class AAA {
    public void rideMethod(){System.out.println("AAA's Method");}
    public void loadMethod(){System.out.println("void Method");}
}
class BBB extends AAA {
    public void rideMethod(){System.out.println("BBB's Method");}
    public void loadMethod(int num){System.out.println("int Method");}
}
class CCC extends BBB {
    public void rideMethod(){System.out.println("CCC's Method");}
    public void loadMethod(double num){System.out.println("double Method");}
}
class RideAndLoad {
    public static void main(String[] args) {
        AAA ref1 = new CCC();
        BBB ref2 = new CCC();
        CCC ref3 = new CCC();

        ref1.rideMethod();
        ref2.rideMethod();
        ref3.rideMethod();

        ref3.loadMethod();
        ref3.loadMethod(1);
        ref3.loadMethod(1.2);
    }
}
```

```
CCC's Method
CCC's Method
CCC's Method
void Method
int Method
double Method
```



☞ 인스턴스 변수도 오버라이딩 되는가?

```
class AAA  {
    public int num=2;
}

class BBB extends AAA {
    public int num=5;
}

class CCC extends BBB {
    public int num=7;
}

class ValReDecle {
    public static void main(String[] args) {
        CCC ref1 = new CCC();
        BBB ref2 = ref1;
        AAA ref3 = ref2;

        System.out.println("CCC's ref: "+ref1.num);
        System.out.println("BBB's ref: "+ref2.num);
        System.out.println("AAA's ref: "+ref3.num);
    }
}
```



✓ 인스턴스 변수는 오버라이딩 관계에 놓이지 않는다. 따라서 참조변수의 자료형에 따라서 접근대상이 결정된다.

```
CCC's ref : 7
BBB's ref : 5
AAA's ref : 2
```

☞ instanceof 연산자

- 형변환이 가능한지를 묻는 연산자이다.
- 형변환이 가능하면 true를 가능하지 않으면 false를 반환.

```
class Box
{
    public void simpleWrap()
    { System.out.println("simple wrap"); }
}

class PaperBox extends Box
{
    public void paperWrap()
    { System.out.println("paper wrap"); }
}

class GoldPaperBox extends PaperBox
{
    public void goldWrap()
    { System.out.println("gold wrap"); }
}
```

```
class InstanceOf {
    public static void wrapBox(Box box) {
        if(box instanceof GoldPaperBox)
            ((GoldPaperBox)box).goldWrap();
        else if(box instanceof PaperBox)
            ((PaperBox)box).paperWrap();
        else
            box.simpleWrap();
    }
    public static void main(String[] args)
    {
        Box box1=new Box();
        PaperBox box2=new PaperBox();
        GoldPaperBox box3=new GoldPaperBox();

        wrapBox(box1);
        wrapBox(box2);
        wrapBox(box3);
    }
}
```

box가 GoldPaperBox로 형변환 가능하다면

☞ 개인 정보 관리 프로그램

```
import java.util.Scanner;

class Friend {
	String name;
	String phoneNum;
	String addr;

	public Friend(String name, String phone, String addr) {
		this.name=name;
		this.phoneNum=phone;
		this.addr=addr;
	}

	public void showData() {
		System.out.println("이름 : "+name);
		System.out.println("전화 : "+phoneNum);
		System.out.println("주소 : "+addr);
	}

	public void showBasicInfo(){ }
}
```

```
class HighFriend extends Friend {    // 고교동창
    String work;

    public HighFriend(String name, String phone, String addr, String job) {
        super(name, phone, addr);
        work=job;
    }

    public void showData() {
        super.showData();
        System.out.println("직업 : "+work);
    }

    public void showBasicInfo() {
        System.out.println("이름 : "+name);
        System.out.println("전화 : "+phoneNum);
    }
}
```

```
class UnivFriend extends Friend {    // 대학동기
    String major;                    // 전공학과

    public UnivFriend(String name, String phone, String addr, String major) {
        super(name, phone, addr);
        this.major=major;
    }

    public void showData() {
        super.showData();
        System.out.println("전공 : "+major);
    }

    public void showBasicInfo() {
        System.out.println("이름 : "+name);
        System.out.println("전화 : "+phoneNum);
        System.out.println("전공 : "+major);
    }
}
```

```
class FriendInfoHandler {

	private Friend[] myFriends;
	private int numOfFriends;

	public FriendInfoHandler(int num) {
		myFriends=new Friend[num];
		numOfFriends=0;
	}

	private void addFriendInfo(Friend fren) {
		myFriends[numOfFriends++]=fren;
	}

	public void addFriend(int choice) {
		String name, phoneNum, addr, job, major;

		Scanner sc = new Scanner(System.in);
		System.out.print("이름 : "); name = sc.nextLine();
		System.out.print("전화 : "); phoneNum = sc.nextLine();
		System.out.print("주소 : "); addr = sc.nextLine();

		if(choice==1) {
			System.out.print("직업 : "); job=sc.nextLine();
			addFriendInfo(new HighFriend(name, phoneNum, addr, job));
		} else {  // if(choice==2)

			System.out.print("학과 : "); major=sc.nextLine();
			addFriendInfo(new UnivFriend(name, phoneNum, addr, major));
		}

		System.out.println("입력 완료! ₩n");
	}
```

```
	public void showAllData()
	{
		for(int i=0; i<numOfFriends; i++)
		{
			myFriends[i].showData();
			System.out.println("");
		}
	}

	public void showAllSimpleData()
	{
		for(int i=0; i<numOfFriends; i++)
		{
			myFriends[i].showBasicInfo();
			System.out.println("");
		}
	}
}
```

```
class MyFriendInfoBook {
	public static void main(String[] args) {
		FriendInfoHandler handler = new FriendInfoHandler(10);

		while(true) {
			System.out.println("*** 메뉴 선택 ***");
			System.out.println("1. 고교 정보 저장");
			System.out.println("2. 대학 친구 저장");
			System.out.println("3. 전체 정보 출력");
			System.out.println("4. 기본 정보 출력");

			System.out.println("5. 프로그램 종료");
			System.out.print("선택   >> ");

			Scanner sc = new Scanner(System.in);
			int choice = sc.nextInt();

			switch(choice) {
			case 1: case 2:
				handler.addFriend(choice);
				break;
			case 3:
				handler.showAllData();
				break;
			case 4:
				handler.showAllSimpleData();
				break;
			case 5:
				System.out.println("프로그램을 종료합니다.");
				return;
			}
		}
	}
}
```

☞ 다음 클래스 정의에서 상속의 이유를 찾아보자.

class HighFriend extends Friend

class UnivFriend extends Friend

- Friend 클래스는 인스턴스화 되지 않는다.
- 다만, HighFriend 클래스와 UnivFriend 클래스의 상위 클래스로만 의미를 지닌다.
- Friend 클래스의 showBasicInfo 메소드를 하위 클래스에서 각각 오버라이딩 하고 있다.
- Friend 클래스의 showBasicInfo 메소드는 비어있다!

그렇다면! 굳이 Friend 클래스를 정의하면서까지 HighFriend 클래스와UnivFriend 클래스를 상속의 관계에 두는 이유는 무엇인가! 이 문제에 대한 답은 FriendInfoHandler 클래스에서 찾을 수 있다.

☞ FriendInfoHandler 클래스의 관찰

```
class FriendInfoHandler
{
    private Friend[] myFriends;
    private int numOfFriends;

    public FriendInfoHandler(int num)
    {
        myFriends=new Friend[num];
        numOfFriends=0;
    }

    private void addFriendInfo(Friend fren)
    {
        myFriends[numOfFriends++]=fren;
    }
    . . . . . .
    public void showAllSimpleData()
    {
        for(int i=0; i<numOfFriends; i++)
        {
            myFriends[i].showBasicInfo();
            System.out.println("");
        }
    }
}
```

✓ Friend 클래스를 상속했기 때문에 Friend의 하위 클래스의 인스턴스가 저장 가능하다!

➢ FriendInfoHandler 클래스는 Friend의 하위 클래스의 인스턴스를 저장 및 관리한다.

➢ FriendInfoHander 클래스 입장에서는 HighFriend 클래스의 인스턴스도, UnivFriend 클래스의 인스턴스도 모두 Friend 클래스의 인스턴스로 간주한다.

✓ showBasicInfo 메소드를 오버라이딩 했기 때문에 Friend 클래스의 참조변수를 통해서도 하위클래스의 showBasicInfo 메소드를 호출할 수 있다! 이것이 바로 showBasicInfo 메소드를 오버라이딩의 관계에 둔 이유이다!

☞ showBasicInfo 메소드의 오버라이딩 이유

- showBasicInfo 메소드를 오버라이딩 관계에 두지 않는다면 FriendInfoHandler 클래스의 showAllSimpleData 메소드는 다음과 같이 변경되어야 한다.

```
void showAllSimpleData( )
{
   for(int i=0; i<numOfFriends; i++)
   {
      if(myFriends[i] instanceof HighFriend)
         ((HighFriend)myFriends[i]).showBasicInfo();
      else
         ((UnivFriend)myFriends[i]).showBasicInfo();

      System.out.println("");
   }
}
```

변경된 부분

그러나 이것이 전부가 아니다. Friend 클래스를 상속하는 하위클래스가 하나 더 등장할 때 마다 위의 메소드는 엄청나게 복잡해진다. 특히 UnivFriend 클래스와 HighFriend 클래스를 상속의 관계로 묶지 않았다면, FriendInfoHandler 클래스는 지금보다 훨씬 더 복잡해져야 하며, Friend 클래스를 상속하는 하위클래스의 수가 증가할 때 마다 엄청난 코드의 확장이 필요해진다.

☞ 상속과 오버라이딩이 가져다 주는 이점

"상속을 통해 연관된 일련의 클래스에 대한 공통적인 규약을 정의할 수 있습니다."

위의 문장은 Friend 관련 예제를 통해서 다음과 같이 이해되어야 한다!

FriendInfoHandler 클래스는, 상속을 통해 연관된 HighFriend, UnivFriend 클래스에 대해(일련의 클래스에 대해) 동일한 방식으로 배열에 저장 및 메소드 호출을 할 수(공통적인 규약을 정의할 수) 있다.

☞ 모든 클래스는 Object 클래스를 상속한다.

```
class MyClass { . . . }

class MyClass extends Object { . . . }
```

✓ 자바 클래스가 아무것도 상속하지 않으면 java.lang 패키지의 Object 클래스를 자동으로 상속한다. 때문에 모든 자바클래스는 Object 클래스를 직접적 혹은 접적으로 상속한다.

```
Object obj1=new MyClass();
Object obj2=new int[5];   // 배열도 인스턴스이므로 작성 가능
```

- 모든 클래스가 Object 클래스를 직접 혹은 간접적으로 상속하므로, 다음 두 가지가 가능하다.
  - 자바의 모든 인스턴스는 Object 클래스의 참조변수로 참조 가능
  - 자바의 모든 인스턴스를 대상으로 Object 클래스에 정의된 메소드 호출 가능

☞ 클래스의 final 선언과 final 메소드

```
final class MyClass
{
    . . . .
}
```

클래스 MyClass가 상속되는 것을 허용하지 않는다!

```
class YourClass
{
    final void yourFunc(int n) { . . . }
    . . . .
}
```

클래스 YourClass가 상속되는 것을 허용은 하되, 메소드 yourFunc의 오버라이딩은 허용하지 않는다!

대표적인 final 메소드로는 Object 클래스의 wait, notify, notifyall 메소드 등이 있으며, 이들은 실제로 오버라이딩이 바람직하지 않은 메소드들이다.

abstract 클래스 & interface & inner class

☞ abstract 클래스

➢ 인스턴스의 생성을 허용 안 하는 abstract 클래스

```
abstract class Friend// 하나 이상의 메소드가 abstract면, 클래스도 abstract
{
        ……
        public void showData()
        {
            System.out.println("이름 : "+name);
            System.out.println("전화 : "+phoneNum);
            System.out.println("주소 : "+addr);
        }
        public abstract void showBasicInfo();// 메소드를 완성시키지 않는다는 선언
}
```

- showBasicInfo 메소드는 비어 있었다. 이렇듯 오버라이딩의 관계유지를 목적으로 하는 메소드는 abstract로 선언이 가능하다.
- 하나 이상 abstarct 메소드를 포함하는 클래스는 abstract로 선언되어야 하며, 인스턴스 생성은 불가!
- 인스턴스 생성은 불가능하나, 참조 변수 선언 가능하고, 오버라이딩의 원리 그대로 적용됨!

☞ abstract 클래스를 상속하는 하위 클래스

```
abstarct class AAA
{
    void methodOne() { . . . }
    abstract void methodTwo();
}

class BBB extends AAA
{
    void methodThree() { . . . }
}
```

그대로 하위클래스로 내려오는 꼴이 된다!

컴파일 에러 발생. BBB 클래스도 abstract로 선언되어야 에러 발생 않는다!

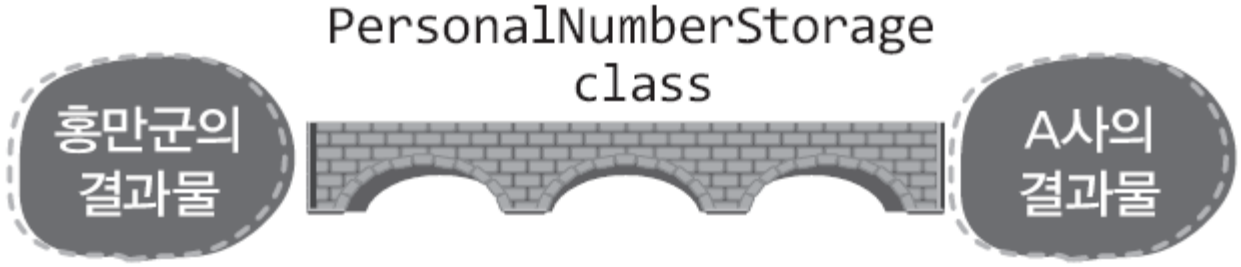
위의 경우 BBB 클래스는 AAA 클래스의 abstract 메소드를 상속한다. 그런데 오버라이딩 하지 않았으므로, abstract 상태 그대로 놓이게 된다. 결국 BBB 클래스는 하나 이상의 abstract 메소드를 포함한 셈이니, abstract로 선언되어야 하며, 인스턴스의 생성도 불가능하게 된다.

☞ interface

```
abstract class PersonalNumberStorage
{
    public abstract void addPersonalInfo(String perNum, String name);
    public abstract String searchName(String perNum);
}


class AbstractInterface
{
    public static void main(String[] args)
    {
        PersonalNumberStorage storage=new (A사가 구현할 클래스 이름);
        storage.addPersonalInfo("김기동", "950000-1122333");
        storage.addPersonalInfo("장산길", "970000-1122334");

        System.out.println(storage.searchName("950000-1122333"));
        System.out.println(storage.searchName("970000-1122334"));
    }
}
```

클래스 PersonalNumberStorage의 정의로 인해서 윗쪽의 형태로 프로젝트를 완료 할 수 있게 되었다. A사의 프로젝트 진행에 상관 없이 말이다!

☞ 인터페이스에 대한 고찰



- 클래스 PersonalNumberStorage는 인터페이스의 역할을 하는 클래스 이다.
- 인터페이스는 두 결과물의 연결고리가 되는 일종의 약속 역할을 한다.
- 인터페이스의 정의로 인해서 홍만군은 홍만군 대로, A사는 A사대로 더 이상의 추가 논의 없이 프로젝트를 진행할 수 있었다.
- 인터페이스의 정의되었기 때문에 프로젝트를 하나로 묶는 과정도 문제가 되지 않는다.

☞ interface의 활용

```
abstract class PersonalNumberStorage
{
    public abstract void addPersonalInfo(String perNum, String name);
    public abstract String searchName(String perNum);
}
```

```
interface PersonalNumberStorage
{
    void addPersonalInfo(String perNum, String name);
    String searchName(String perNum);
}
```

- 인터페이스내에 선언된 변수는 무조건 public static final로 선언된다.
- 인터페이스내에 선언된 메소드는 무조건 public abstract로 선언된다.
- 인터페이스도 참조 변수 선언가능 하고, 메소드 오버라이딩 원칙 그대로 적용된다!

☞ interface의 특성

```
public interface MyInterface
{
    public void myMethod();
}

public interface YourInterface
{
    public void yourMethod();
}
```

```
public interface SuperInterf
{
    public void supMethod();
}

public interface SubIntef extends SuperInterf
{
    public void subMethod();
}
```

인터페이스 간 상속 가능. 단 이때는 implements가 아닌 extends를 사용한다.

```
Class OurClass implements MyInterface, YourInterface
{
    public void myMethod() { . . . }
    public void yourMethod() { . . . }
}
```

인터페이스는 둘 이상을 동시에 구현 가능.

인터페이스의 상속(구현)은 extends가 아닌 implements를 사용한다.

☞ 자바 interface의 또 다른 가치

```
interface UpperCasePrintable
{
    // 비어 있음
}

class ClassPrinter
{
    public static void print(Object obj)
    {
        String org=obj.toString();
        if(obj instanceof UpperCasePrintable)
        {
            org=org.toUpperCase();
        }

        System.out.println(org);
    }
}
```

UpperCasePrintable의 성격을 표시하는 용도!

```
class PointOne implements UpperCasePrintable
{
    private int xPos, yPos;

    PointOne(int x, int y)
    {
        xPos=x;
        yPos=y;
    }
    public String toString()
    {
        String posInfo="[x pos : "+xPos + ", y pos : "+yPos+"]";
        return posInfo;
    }
}
```

- 무엇인가를 표시하는(클래스의 특성을 표시하는)용도로도 인터페이스는 사용된다.
- 이러한 경우, 인터페이스의 이름은 ~able로 끝나는 것이 보통이다.
- 이러한 경우, 인터페이스는 비어 있을 수도 있다.
- instanceof 연산자를 통해서 클래스의 특성을 파악할 수 있다.

☞ Inner 클래스와 Nested 클래스

```
class OuterClass
{
    . . . .
    class InnerClass
    {
        . . . .
    }
}
```

클래스의 정의가 다른 클래스의 내부에 삽입될 수 있다. 이때 외부의 클래스를 가리켜 Outer 클래스라 하고, 내부의 클래스를 가리켜 Inner 클래스라 한다.

```
class OuterClass
{
    . . . .
    static class NestedClass
    {
        . . . .
    }
}
```

Inner 클래스의 형태에 static 선언이 삽입되면, 이를 가리켜 static Inner 클래스 또는 간단히 Nested 클래스라 한다.

☞ Nested 클래스의 이해

```
class OuterClassOne
{
    OuterClassOne()
    {
        NestedClass nst=new NestedClass();
        nst.simpleMethod();
    }
    static class NestedClass
    {
        public void simpleMethod()
        {
            System.out.println("Nested Instance Method One");
        }
    }
}
```

클래스 내부에서는 직접 생성 가능

클래스 외부에서 이 클래스의 이름은 OuterClassOne.NestedClass가 된다!

```
public static void main(String[] args)
{
    OuterClassOne one=new OuterClassOne();

    OuterClassOne.NestedClass nst1=new OuterClassOne.NestedClass();
    nst1.simpleMethod();
}
```

Nested 클래스의 인스턴스 생성 방법

NestedClass가 private으로 선언되면, 선언된 클래스 내부에서만 인스턴스를 생성할 수 있다.

☞ Inner 클래스의 이해

```
class OuterClass
{
    private String myName;
    private int num;

    OuterClass(String name)
    {
        myName=name;
        num=0;
    }

    public void whoAreYou()
    {
        num++;
        System.out.println(myName+ " OuterClass "+num);
    }

    class InnerClass
    {
        InnerClass()
        {
            whoAreYou();
        }
    }
}
```

```
public static void main(String[] args)
{
    OuterClass out1=new OuterClass("First");
    OuterClass out2=new OuterClass("Second");
    out1.whoAreYou();
    out2.whoAreYou();

    OuterClass.InnerClass inn1=out1.new InnerClass();
    OuterClass.InnerClass inn2=out2.new InnerClass();
    OuterClass.InnerClass inn3=out1.new InnerClass();
    OuterClass.InnerClass inn4=out1.new InnerClass();
    OuterClass.InnerClass inn5=out2.new InnerClass();
}
```

Inner 클래스의 인스턴스는
Outer 클래스의 인스턴스에 종속적이다!

Inner 클래스의 성격

- Outer 클래스의 인스턴스 생성 후에야 Inner 클래스의 인스턴스 생성이 가능하다.
- Inner 클래스 내에서는 Outer 클래스의 멤버에 직접 접근이 가능하다.
- Inner 클래스의 인스턴스는 자신이 속할 Outer 클래스의 인스턴스를 기반으로 생성된다.

☞ Local 클래스

- Outer 클래스의 인스턴스 생성 후에야 Inner 클래스의 인스턴스 생성이 가능하다.
- Inner 클래스 내에서는 Outer 클래스의 멤버에 직접 접근이 가능하다.
- Inner 클래스의 인스턴스는 자신이 속할 Outer 클래스의 인스턴스를 기반으로 생성된다.

Inner 클래스의 성격을 그대로 유지한다!

Local 클래스는 메소드 내에 정의가 되어서, 메소드 내에서만 인스턴스의 생성 및 참조 변수의 선언이 가능하다는 특징이 있다!

```
class OuterClass
{
   . . . .
   public LocalClass createLocalClassInst( )
   {
      class LocalClass
      {
         . . . .
      }

      return new LocalClass( );
   }
}
```

문제 없는 반환형인가?

Local 클래스

왼쪽에 정의된 LocalClass 클래스는 로컬 클래스이다! 그러나 반환형의 선언이 문제가 된다. 반환하는 로컬클래스를 외부에서 참조할 수 없기 때문이다. 참조 변수의 선언은 클래스가 정의된 메소드 내에서만 가능하므로...

☞ Local 클래스의 적절한 사용 모델

```
interface Readable
{
    public void read();
}

class OuterClass
{
    private String myName;

    OuterClass(String name)
    {
        myName=name;
    }
    public Readable createLocalClassInst()
    {
        class LocalClass implements Readable
        {
            public void read()
            {
                System.out.println("Outer inst name : "+myName);
            }
        }
        return new LocalClass();
    }
}
```

```
public static void main(String[] args)
{
    OuterClass out1=new OuterClass("First");
    Readable localInst1=out1.createLocalClassInst();
    localInst1.read();

    OuterClass out2=new OuterClass("Second");
    Readable localInst2=out2.createLocalClassInst();
    localInst2.read();
}
```

이렇게 인터페이스의 구현을 기반으로 로컬클래스를 정의하면 외부에 정의된 인터페이스의 참조변수를 통해서 인스턴스의 참조가 가능하다!

☞ Local 클래스의 지역변수, 매개변수 접근

```
public Readable createLocalClassInst(final int instID)
{
    class LocalClass implements Readable
    {
        public void read()
        {
            System.out.println("Outer inst name : "+myName);
            System.out.println("Local inst ID : "+instID);
        }
    }
    return new LocalClass();
}
```

- 메소드가 반환하는 순간 매개변수와 지역변수는 소멸된다.
- 따라서 매개변수와 지역변수의 접근은 논리적으로 맞지 않는다!
- 단, final로 선언이 변수의 접근은 허용한다.
- 접근의 허용을 위해서 final 변수를 로컬클래스의 인스턴스가 접근 가능한 영역에 복사한다.

☞ Anonymous 클래스

클래스의 이름이 정의되어 있지 않다는 사실에서만 Local 클래스와 차이를 보인다!

```
public Readable createLocalClassInst(final int instID)
{
    return new Readable()
    {
        public void read()
        {
            System.out.println("Outer inst name : "+myName);
            System.out.println("Local inst ID : "+instID);
        }
    };
}
```