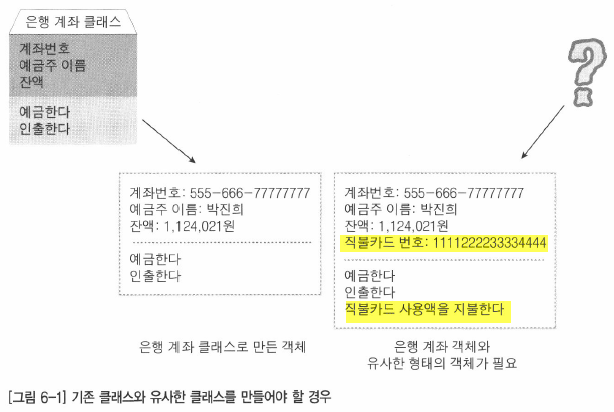
**Inheritance / Interface / Abstraction / Polymerphism**

**I. Inheritance**

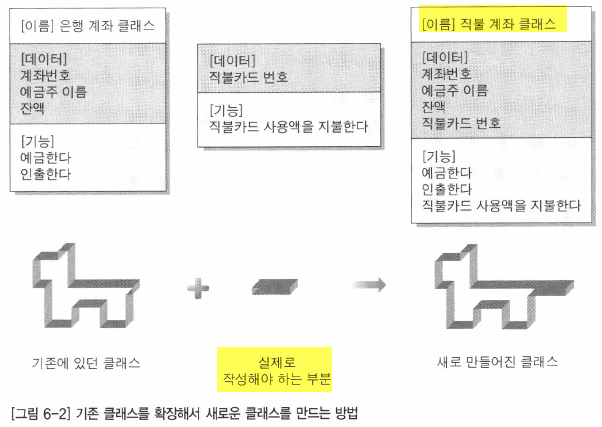
**01. Class의 상속**

(1) 상속의 개념

기존의 객체와 아주 유사한 특성의 객체가 필요한 경우 그 객체를 새로 만들기 보다는 유사한 클래스(소스코드 수정하지 않고)를 가져다가 필요한 부분만 추가해서 새로운 클래스를 만드는 개념



기존의 클래스가 가지고 있는 데이터구조와 기능은 그대로 물려 받고 새로운 테이터 구조와 기능을 추가해서 새롭게 클래스를 만드는 기술을 상속이라고 한다.



(2) 다른 클래스를 상속하는 클래스 선언하기

|  |  |
| --- | --- |
| 은행클래스  (기존,Super Class) | **package** p01.Inheritance;  **public** **class** Account {  String accountNo;//계좌번호  String ownerName;//예금주 이름  **int** balance;//잔액    **void** deposit(**int** amount){//예금  balance += amount;  }  **int** withdraw(**int** amount) **throws** Exception{//인출  **if**(balance < amount)//잔액이 요구금액보다 적으면 Exception발생  **throw** **new** Exception("잔액이 부족합니다.");  balance -= amount;  **return** amount;  }  } |
| 직불계좌클래스  (추가,SubClass) | **package** p01.Inheritance;  **public** **class** CheckingAccount **extends** Account {  String cardNo; //직불카드 번호에 해당하는 필드 선언  //직불카드 사용액을 지불한다에 해당하는 메소드 선언  **int** pay(String cardNo, **int** amount) **throws** Exception{  **if**(!cardNo.equals(**this**.cardNo) || (balance < amount))  **throw** **new** Exception("지불이 불가능합니다.");  **return** withdraw(amount);  }  } |
|  |
| 직불교통계좌클래스 | 직불계좌 클래스를 상속받음.  직불계좌클래스에 선언된 필드,메소드 뿐만 아니라 Super Class인 은헁계좌클래스로부터 상속받은 필드, 메소드까지도 상속받음. |

(3) 다른 클래스를 상속하는 클래스의 사용방법

|  |
| --- |
| **package** p01.Inheritance;  **public** **class** InheritanceEx01 {  **public** **static** **void** main(String[] args) {  CheckingAccount obj = **new** CheckingAccount();  obj.accountNo = "111-22-33333";//Account 상속받은 필드  obj.ownerName = "홍길동";  obj.cardNo = "5555-6666-7777-8888";  obj.deposit(100000);//Account 상속받은 메소드  **try** {  **int** paidAmount = obj.pay("5555-6666-7777-8888", 47000);  System.***out***.println("지불액: " + paidAmount);  System.***out***.println("잔액: " + obj.balance);  } **catch** (Exception e) {  String msg = e.getMessage();  System.***out***.println(msg);  }  }  } |
| 지불액: 47000  잔액: 53000 |

(4) 다른 클래스를 상속하는 클래스에 생성자를 선언하는 방법

: Super Class로부터 상속받은 필드들도 생성자안에서 초기화 해야한다.

|  |
| --- |
| **package** p02.Inheritance;  **public** **class** CheckingAccount **extends** Account {  String cardNo;  CheckingAccount(String accountNo, String ownerName, **int** balance,String cardNo){//생성자  **this**.accountNo = accountNo;//슈퍼클래스로 상속받은 필드들을 초기화  **this**.ownerName = ownerName;  **this**.balance = balance;  **this**.cardNo = cardNo;//클래스안에 선언된 필드를 초기화  }  **int** pay(String cardNo, **int** amount) **throws** Exception {  **if** (!cardNo.equals(**this**.cardNo) || (balance < amount))  **throw** **new** Exception("지불이 불가능합니다.");  **return** withdraw(amount);  }  } |
| **package** p02.Inheritance;  **public** **class** InheritanceEx02 {  **public** **static** **void** main(String[] args) {  //CheckingAccount의 생성자 호출  CheckingAccount obj = **new** CheckingAccount("111-22-33333","홍길동",0,"5555-6666-7777-8888");  obj.deposit(100000);//Account 상속받은 메소드  **try** {  **int** paidAmount = obj.pay("5555-6666-7777-8888", 47000);  System.***out***.println("지불액: " + paidAmount);  System.***out***.println("잔액: " + obj.balance);  } **catch** (Exception e) {  String msg = e.getMessage();  System.***out***.println(msg);  }  }  } |
| 지불액: 47000  잔액: 53000 |

(5) 생성자가 있는 슈퍼클래스를 상속하는 방법

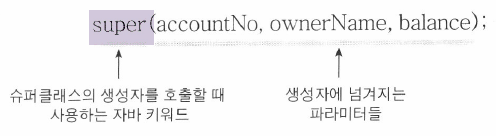
① 에러가 발생한 경우

|  |
| --- |
| //은행 계좌 (기존,부모)  **package** p03.Inheritance;  **public** **class** Account {  String accountNo;//계좌번호  String ownerName;//예금주 이름  **int** balance;//잔액  **public** Account(String accountNo, String ownerName, **int** balance) {//생성자추가  **this**.accountNo = accountNo;  **this**.ownerName = ownerName;  **this**.balance = balance;  }  **public** **void** deposit(**int** amount){//예금  balance += amount;  }  //throws Exception: 메소드가 발생 익셉션의 종류를 표시하는 throws절  **public** **int** withdraw(**int** amount) **throws** Exception{//인출  **if**(balance < amount)//잔액이 요구금액보다 적으면 Exception발생  **throw** **new** Exception("잔액이 부족합니다.");  // return 0;  balance -= amount;  **return** amount;  }  } |
| **package** p02.Inheritance;  **public** **class** CheckingAccount **extends** Account {  String cardNo;  CheckingAccount(String accountNo, String ownerName, **int** balance, String cardNo){  **this**.accountNo = accountNo;//슈퍼클래스로 상속받은 필드들을 초기화  **this**.ownerName = ownerName;  **this**.balance = balance;  **this**.cardNo = cardNo;//클래스안에 선언된 필드를 초기화  }  **int** pay(String cardNo, **int** amount) **throws** Exception {  **if** (!cardNo.equals(**this**.cardNo) || (balance < amount))  **throw** **new** Exception("지불이 불가능합니다.");  **return** withdraw(amount);  }  } |
| [컴파일 에러발생]  "Account( )라는 생성자를 찾을수 없다" : 자바 컴파일러가 컴파일 할때 생성자의 첫번째 명령문이 super class의 생성자 호출문이 아니면 자동으로 슈퍼클래스의 파라메터없는 생성자를 그 위치에 추가하기 때문 |

[해결방법 2가지] :

* **Account 클래스에 파라메터 없는 생성자 추가 : 상속위배- 소스코드를 건드리지 않고 사용(비추천)**
* **CheckingAccount 클래스의 생성자안에 슈퍼클래스의 생성자 호출문을 명시적으로 기재(생성자안에 생성자의 호출문이 있으면, 파라메터없는 호출문을 추가하지 않기 때문이다)**

※ Super Class 호출문 작성 방법



② 정상

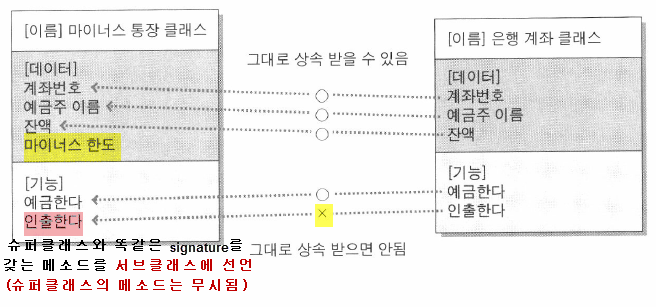
|  |
| --- |
| //은행 계좌 (기존,부모)  **package** p03.Inheritance;  **public** **class** Account {  String accountNo;//계좌번호  String ownerName;//예금주 이름  **int** balance;//잔액  **public** Account(String accountNo, String ownerName, **int** balance) {//생성자  **this**.accountNo = accountNo;  **this**.ownerName = ownerName;  **this**.balance = balance;  }  **public** **void** deposit(**int** amount){//예금  balance += amount;  }  //throws Exception: 메소드가 발생 익셉션의 종류를 표시하는 throws절  **public** **int** withdraw(**int** amount) **throws** Exception{//인출  **if**(balance < amount)//잔액이 요구금액보다 적으면 Exception발생  **throw** **new** Exception("잔액이 부족합니다.");  // return 0;  balance -= amount;  **return** amount;  }  } |
| **package** p03.Inheritance;  **public** **class** CheckingAccount **extends** Account {  String cardNo;  **public** CheckingAccount(String accountNo, String ownerName, **int** balance,String cardNo) {  **super**(accountNo, ownerName, balance);//슈퍼클래스의 생성자 호출문  **this**.cardNo = cardNo;//클래스안에 선언된 필드를 초기화  }  **int** pay(String cardNo, **int** amount) **throws** Exception {  **if** (!cardNo.equals(**this**.cardNo) || (balance < amount))  **throw** **new** Exception("지불이 불가능합니다.");  **return** withdraw(amount);  }  } |

**II. Inherit Method Overriding**

직불계좌 클래스는 기존 은행 계좌 클래스에 새로운 데이터 구조와 기능을 추가 했었지만, 기존의 기능을 바꾸는 일도 할수 있다.

① 슈퍼클래스와 똑같은 signature를 갖는 메소드를 SubClass에 선언

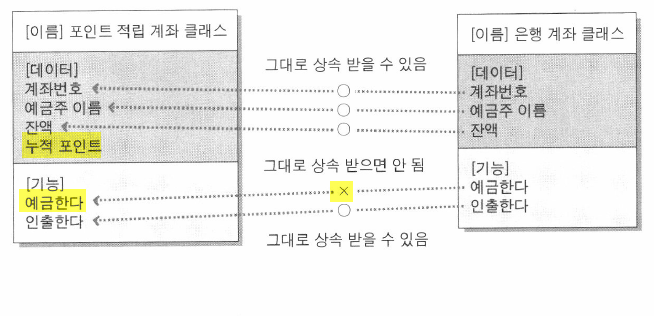
은행 계좌 클래스를 상속받아 "마이너스 통장 클래스"를 만든다고 했을때



|  |
| --- |
| //은행 계좌 (기존,부모)  **package** p04.Inheritance.ok;  **public** **class** Account {  String accountNo;//계좌번호  String ownerName;//예금주 이름  **int** balance;//잔액  **public** Account(String accountNo, String ownerName, **int** balance) {  **this**.accountNo = accountNo;  **this**.ownerName = ownerName;  **this**.balance = balance;  }  **public** **void** deposit(**int** amount){//예금  balance += amount;  }    **public** **int** withdraw(**int** amount) **throws** Exception{// 슈퍼클래스의 메소드 무시됨  **if**(balance < amount)  **throw** **new** Exception("잔액이 부족합니다.");  // return 0;  balance -= amount;  **return** amount;  }  } |
| **package** p05.Inheritance.Overriding;  **public** **class** CreditLineAccount **extends** Account {  **int** creditLine;  **public** CreditLineAccount(String accountNo, String ownerName, **int** balance, **int** creditLine) {  **super**(accountNo, ownerName, balance);  **this**.creditLine = creditLine;  }  //인출한다 기능을 다시 구현하는 메소드 (Method Overiding)  **public** **int** withdraw(**int** amount) **throws** Exception{  **if**((balance + creditLine) < amount)  **throw** **new** Exception("인출이 불가능합니다.");  balance -= amount;  **return** amount;  }  } |
| **package** p05.Inheritance.Overriding;  **public** **class** InheritanceEx03 {  **public** **static** **void** main(String[] args) {  //CheckingAccount의 생성자 호출  CreditLineAccount obj = **new** CreditLineAccount("000-11-11111","홍길동",1000000,2000000);  **try** {  **int** amount = obj.withdraw(47000);  System.***out***.println("인출액: " + amount);  System.***out***.println("잔액: " + obj.balance);  System.***out***.println("마이너스한도: " + obj.creditLine);  } **catch** (Exception e) {  String msg = e.getMessage();  System.***out***.println(msg);  }  }  } |
| 인출액: 47000  잔액: 953000  마이너스한도: 2000000 |

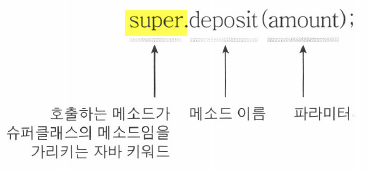
② Override하는 메소드안에서 Override된 메소드 호출하기

은행 계좌 클래스를 상속받아 "포인트 적립 계좌 클래스"를 만든다고 했을때



"예금한다" 기능은 상속 불가 : 예금을 할때마다 포인트를 0.1%씩 적립.

그래서 슈퍼클래스로부터 상속받은 deposit( )를 Override 해서 선언



일단 클래스안에서 그냥 deposit( )를 호출하면 deposit( ) 자신이 호출되기 때문에 호출하는 메소드가 슈퍼클래스의 deposit( )라는 것을 표시해 주어야 한다.

|  |
| --- |
| **package** p05.Inheritance.Overriding;  **public** **class** Account {  String accountNo;//계좌번호  String ownerName;//예금주 이름  **public** **int** balance;//잔액  **public** Account(String accountNo, String ownerName, **int** balance) {  **this**.accountNo = accountNo;  **this**.ownerName = ownerName;  **this**.balance = balance;  }  **public** **void** deposit(**int** amount){//예금  balance += amount;  }  **public** **int** withdraw(**int** amount) **throws** Exception{//인출  **if**(balance < amount)  **throw** **new** Exception("잔액이 부족합니다.");  // return 0;  balance -= amount;  **return** amount;  }  } |
| **package** p05.Inheritance.Overriding;  **public** **class** BonusPointAccount **extends** Account {  **int** bonusPoint;  **public** BonusPointAccount(String accountNo, String ownerName, **int** balance,  **int** bonusPoint) {  **super**(accountNo, ownerName, balance);  **this**.bonusPoint = bonusPoint;  }    **super**.deposit(amount);//SuperClass의 deposit( )호출하여 계산  bonusPoint += (**int**)(amount\*0.001);  }  } |
| **package** p05.Inheritance.Overriding;  **public** **class** InheritanceEx05 {  **public** **static** **void** main(String[] args) {  BonusPointAccount obj = **new** BonusPointAccount("333-33-33333", "김청수",0,0);  obj.deposit(1000000);  System.***out***.println("잔액: " + obj.balance);  System.***out***.println("누적포인트: " + obj.bonusPoint);  }  } |
| 잔액: 2000000  누적포인트: 1000 |

**III. final**

**① final을 이용한 클래스의 상속금지**

|  |
| --- |
| **package** p06.Final;  **final** **class** Account {  String accountNo;//계좌번호  String ownerName;//예금주 이름  **public** **int** balance;//잔액  **public** Account(String accountNo, String ownerName, **int** balance) {  **this**.accountNo = accountNo;  **this**.ownerName = ownerName;  **this**.balance = balance;  }  **public** **void** deposit(**int** amount){//예금  balance += amount;  }  **public** **int** withdraw(**int** amount) **throws** Exception{//인출  **if**(balance < amount)//잔액이 요구금액보다 적으면 Exception발생  **throw** **new** Exception("잔액이 부족합니다.");  balance -= amount;  **return** amount;  }  } |
|  |

**② final을 이용한 Method Overriding 금지**

|  |
| --- |
| **package** p06.Final;  **public** **class** Account {  String accountNo;//계좌번호  String ownerName;//예금주 이름  **public** **int** balance;//잔액  **public** Account(String accountNo, String ownerName, **int** balance) {  **this**.accountNo = accountNo;  **this**.ownerName = ownerName;  **this**.balance = balance;  }  **public** **void** deposit(**int** amount){//예금  balance += amount;  }  **final** **int** withdraw(**int** amount) **throws** Exception{//인출  **if**(balance < amount)//잔액이 요구금액보다 적으면 Exception발생  **throw** **new** Exception("잔액이 부족합니다.");  balance -= amount;  **return** amount;  }  } |
| **package** p06.Final;  **import** p06.Final.Account;  **public** **class** CreditLineAccount **extends** Account {  **public** **int** creditLine;  **public** CreditLineAccount(String accountNo, String ownerName, **int** balance, **int** creditLine) {  **super**(accountNo, ownerName, balance);  **this**.creditLine = creditLine;  }  //인출한다 기능을 다시 구현하는 메소드  **public** **int** withdraw(**int** amount) **throws** Exception{  **if**((balance + creditLine) < amount)  **throw** **new** Exception("인출이 불가능합니다.");  balance -= amount;  **return** amount;  }  } |

=> Account.java에서 final을 제거하면, 정상

**IV. abstract**

: 클래스를 가지고 객체를 만드는 일을 인스턴스화라고 한다. 어떤 경우에 있어서는 인스턴스화를 막아야할 경우가 있다. 그때 사용되는 것이 abstract이다.

**(1) abstract 를 이용한 클래스의 인스턴스화 금지**

예를들면, 은행 계좌 클래스,직불 계좌 클래스, 마이너스 통장 클래스, 포인트 적립 클래스를 사용하는 은행이

정책이 변경되어서 보통 예금 계좌를 개설하지 않기도 되었을때(은행계좌 클래스 객체를 더이상 생성하면 안될때) Account 클래스는 다른 클래스의 Super Class이기 때문에 인스턴화는 막더라도 없애버리면 안된다.

이럴때, Account 클래스의 선언 제일 앞에 abstract라는 키워드를 사용한다.

이와같이 abstract를 붙인 클래스를 추상클래스라 한다. 객체를 만들수 없으므로 에러 발생

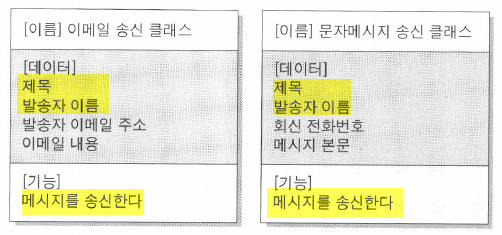
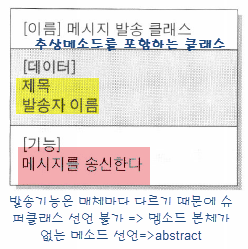
|  |
| --- |
| **package** p07.Abstract;  **abstract** **class** Account {  String accountNo;//계좌번호  String ownerName;//예금주 이름  **public** **int** balance;//잔액  **public** Account(String accountNo, String ownerName, **int** balance) {  **this**.accountNo = accountNo;  **this**.ownerName = ownerName;  **this**.balance = balance;  }  **public** **void** deposit(**int** amount){//예금  balance += amount;  }  **int** withdraw(**int** amount) **throws** Exception{//인출  **if**(balance < amount)//잔액이 요구금액보다 적으면 Exception발생  **throw** **new** Exception("잔액이 부족합니다.");  balance -= amount;  **return** amount;  }  } |
| **package** p07.Abstract;  **public** **class** InheritanceEx {  **public** **static** **void** main(String[] args) {  Account obj = **new** Account("111-222-3333","오바마",0);  }  } |

=>abstract로 인해서 컴파일 에러

**(2) Method 본체가 없는 추상 메소드**

: 프로그램 개발 때부터 추상 클래스로 선언해야 할 경우, 예를들어 고객에게 일괄적으로 메시지를 발송하는 프로그램을 작성하다가 (이메일 송신 클래스, 문자메시지 송신 클래스), 이 두 클래스는 데이터구조와 기능에 공통점을 뽑아 슈퍼클래스를 만들었다.

추상클래스안에서 추상메서드는 본체가 없기 때문에 서블 클래스에 상속해줄 로직도 없고,객체를 만들수 없기 때문에 직접 호출할수도 없다.

추상메소드 선언 : 추상메소드를 포함하는 클래스는 클래스 자체도 추상클래스로 선언해야 한다.



|  |
| --- |
| //추상메소드를 포함 하는 메소드 - 메시지 발송 클래스  **package** p07.Abstract;  **abstract** **class** MessageSender {//클래스 자체도 추상클래스로 선언  String title;  String senderName;  **public** MessageSender(String title, String senderName) {  **this**.title = title;  **this**.senderName = senderName;  }  //추상메소드: 메소드 본체가 없음  **abstract** **void** sendMessage(String recipient);  // void sendMessage(String recipient){ //참고용 :일반 메소드  // }  } |
| //메시지 발송 클래스를 상속하는 이메일 송신 클래스  **package** p07.Abstract;  **public** **class** EMailSender **extends** MessageSender {  String senderAddr;  String emailBody;  **public** EMailSender(String title, String senderName, String senderAddr,  String emailBody) {  **super**(title, senderName);  **this**.senderAddr = senderAddr;  **this**.emailBody = emailBody;  }  //Super Class의 메소드를 Override하는 메소드  **void** sendMessage(String recipient) {  System.***out***.println("---------------------------------");  System.***out***.println("제목: " + title);  System.***out***.println("보내는사람: " + senderName + " " + senderAddr);  System.***out***.println("받는 사람: " + recipient);  System.***out***.println("내용: " + emailBody);  }  } |
| //메시지 발송 클래스를 상속하는 문자메시지 송신 클래스  **package** p07.Abstract;  **public** **class** SMSSender **extends** MessageSender {  String returnPhoneNo;  String message;  **public** SMSSender(String title, String senderName, String returnPhoneNo,  String message) {  **super**(title, senderName);  **this**.returnPhoneNo = returnPhoneNo;  **this**.message = message;  }  //Super Class의 메소드를 Override하는 메소드  **void** sendMessage(String recipient) {  System.***out***.println("-------------------------");  System.***out***.println("제목: " + title);  System.***out***.println("보내는 사람: " + senderName);  System.***out***.println("전화번호: " + recipient);  System.***out***.println("회신전화번호: " + returnPhoneNo);  System.***out***.println("메시지 내용: " + message);  }  } |
| //이메일 송신 클래스와 문자 메시지 송신 클래스를 사용하는 프로그램  **package** p07.Abstract;  **public** **class** InheritanceEx06 {  **public** **static** **void** main(String[] args) {  EMailSender obj1 = **new** EMailSender("생일을 축하합니다.", "고객센터",  "admin@gmarket.co.kr", "10%할인쿠폰이 발행되었습니다.");  SMSSender obj2 = **new** SMSSender("생일을 축하 합니다.", "고객센터", "02-999-9999",  "10%할인 쿠폰이 발행 되었습니다.");  obj1.sendMessage("hatman@daum.net");//추상메소드를 구현하는 메소드 호출  obj1.sendMessage("stickman@nave.co.kr");  obj1.sendMessage("010-4434-1111");  }  } |
| ---------------------------------  제목: 생일을 축하합니다.  보내는사람: 고객센터 admin@gmarket.co.kr  받는 사람: hatman@daum.net  내용: 10%할인쿠폰이 발행되었습니다.  ---------------------------------  제목: 생일을 축하합니다.  보내는사람: 고객센터 admin@gmarket.co.kr  받는 사람: stickman@nave.co.kr  내용: 10%할인쿠폰이 발행되었습니다.  ---------------------------------  제목: 생일을 축하합니다.  보내는사람: 고객센터 admin@gmarket.co.kr  받는 사람: 010-4434-1111  내용: 10%할인쿠폰이 발행되었습니다. |

**※ 추상 메소드를 선언하는 목적**

① 슈퍼 클래스에서 추상 메서드로 선언 한것을 서브 클래스에서 이 메서드를 반드시 구현 하도록 만들기 위해서다. 예를들면, 위에서 MessageSender 클래스를 상속하는 SillySender 클래스(서브클래스)가 있다고 할때

|  |
| --- |
| //추상메소드를 포함 하는 메소드 - 메시지 발송 클래스  **package** p07.Abstract;  **abstract** **class** MessageSender {//클래스 자체도 추상클래스로 선언  String title;  String senderName;  **public** MessageSender(String title, String senderName) {  **this**.title = title;  **this**.senderName = senderName;  }  //추상메소드: 메소드 본체가 없음  **abstract** **void** sendMessage(String recipient);  } |
| **package** p07.Abstract;  **public** **class** SillySender **extends** MessageSender {  **public** SillySender(String title, String senderName) {  **super**(title, senderName);  }  //추상메소드가 없어서 컴파일 에러 발생함  } |
| [컴파일 에러]해결방법    1. SillySender 클래스안에서 sendMessage( )를 구현한다. 슈퍼클래스를 Override한다.  2. SillySender 클래스를 추상 클래스로 만든다. |

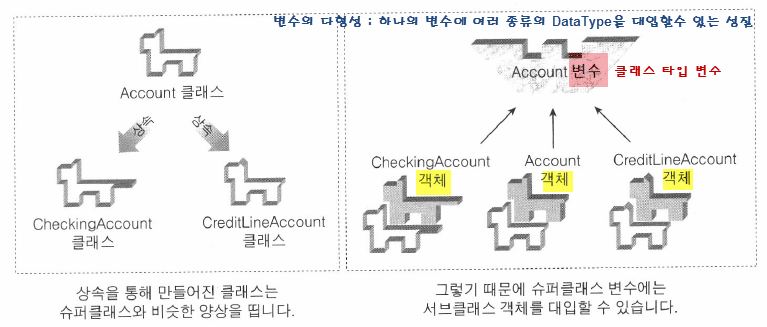
결론 : 추상 메서드를 선언 해두면, 서브클래스의 메서드들이 지켜야할 규칙이 공식적으로 만들어지게 된다.

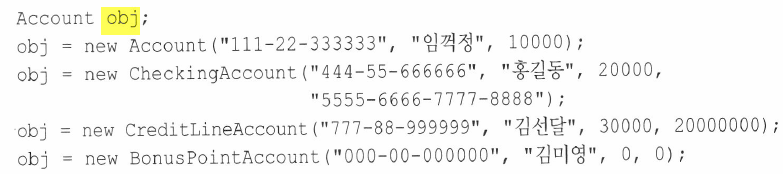
② 다형성

**V. Polymorphism (다형성)**

**01. Class 변수의 다형성**

클래스는 객체를 생성하는 도구지만, 객체의 Type으로도 사용된다. 자바에서 클래스 변수에 그 클래스로 만든 객체만 대입할수 있는게 아니라, 그 클래스의 서브클래스 객체도 대입할수 있다.





⇒ 클래스 변수의 이러한 다형성을 이용하면 여러종류의 객체들을 똑같은 로직으로 처리할수 있다.

|  |
| --- |
| **package** p08.Polymorphism;  **public** **class** Account {  String accountNo;//계좌번호  String ownerName;//예금주 이름  **public** **int** balance;//잔액  **public** Account(String accountNo, String ownerName, **int** balance) {  **this**.accountNo = accountNo;  **this**.ownerName = ownerName;  **this**.balance = balance;  }  **public** **void** deposit(**int** amount){//예금  balance += amount;  }  **public** **int** withdraw(**int** amount) **throws** Exception{//인출  **if**(balance < amount)//잔액이 요구금액보다 적으면 Exception발생  **throw** **new** Exception("잔액이 부족합니다.");  balance -= amount;  **return** amount;  }  } |
| **package** p08.Polymorphism;  **public** **class** CheckingAccount **extends** Account {  String cardNo;  **public** CheckingAccount(String accountNo, String ownerName, **int** balance,String cardNo) {  **super**(accountNo, ownerName, balance);//슈퍼클래스의 생성자 호출문: 반드시 첫번째 호출문이어야한다.  **this**.cardNo = cardNo;//클래스안에 선언된 필드를 초기화  }  **int** pay(String cardNo, **int** amount) **throws** Exception {  **if** (!cardNo.equals(**this**.cardNo) || (balance < amount))  **throw** **new** Exception("지불이 불가능합니다.");  **return** withdraw(amount);  }  } |
| **package** p08.Polymorphism;  **public** **class** CreditLineAccount **extends** Account {  **public** **int** creditLine;  **public** CreditLineAccount(String accountNo, String ownerName, **int** balance, **int** creditLine) {  **super**(accountNo, ownerName, balance);  **this**.creditLine = creditLine;  }  //인출한다 기능을 다시 구현하는 메소드  **public** **int** withdraw(**int** amount) **throws** Exception{  **if**((balance + creditLine) < amount)  **throw** **new** Exception("인출이 불가능합니다.");  balance -= amount;  **return** amount;  }  } |
| **package** p08.Polymorphism;  **public** **class** BonusPointAccount **extends** Account {  **int** bonusPoint;  **public** BonusPointAccount(String accountNo, String ownerName, **int** balance,  **int** bonusPoint) {  **super**(accountNo, ownerName, balance);  **this**.bonusPoint = bonusPoint;  }  **public** **void** deposit(**int** amount){//예금한다 기능을 다시 구현하는 메소드  balance += amount;  **super**.deposit(amount);//SuperClass의 deposit( )호출하여 계산  bonusPoint += (**int**)(amount\*0.001);  }  } |
| **package** p08.Polymorphism;  **public** **class** InheritanceEx07 {  **public** **static** **void** main(String[] args) {  Account obj1 = **new** Account("111-22-3333", "임꺽정", 10000);  CheckingAccount obj2 = **new** CheckingAccount("444-55-666", "홍길동", 20000, "5555-666-777-888");  CreditLineAccount obj3 = **new** CreditLineAccount("777-22-3333", "김선달", 3000, 2000000);  BonusPointAccount obj4 = **new** BonusPointAccount("000-00-00000", "김미영",0, 0);  //다양한 Type의 객체를 가지고 같은 메서드를 호출  *printAccountInfo*(obj1);  *printAccountInfo*(obj2);  *printAccountInfo*(obj3);  *printAccountInfo*(obj4);  }  //다양한 타입의 객체를 한 타입(Account obj)의 파라메터 변수로 받는다.  **static** **void** printAccountInfo(Account obj) {  System.***out***.println("계좌번호: " + obj.accountNo);  System.***out***.println("예금주 이름:" + obj.ownerName);  System.***out***.println("잔액: " + obj.balance);  System.***out***.println();  }  } |
| 계좌번호: 111-22-3333  예금주 이름:임꺽정  잔액: 10000  계좌번호: 444-55-666  예금주 이름:홍길동  잔액: 20000  계좌번호: 777-22-3333  예금주 이름:김선달  잔액: 3000  계좌번호: 000-00-00000  예금주 이름:김미영  잔액: 0 |

**02. Polymorphism & Method Overriding**

subclass들의 객체 슈퍼클래스(추상클래스,추상메서드)의 변수에 대입한다고 해서 그 객체 타입이 슈퍼클래스 타입으로 바뀌는 것은 아니다.

|  |
| --- |
| //추상메소드를 포함 하는 메소드 - 메시지 발송 클래스  **package** p07.Abstract;  **abstract** **class** MessageSender {//클래스 자체도 추상클래스 구성요소  String title; //(1)인스턴스변수  String senderName;  **public** MessageSender(String title, String senderName) { //(2)생성자  **this**.title = title;  **this**.senderName = senderName;  }  //(3) 추상메소드: 메소드 본체가 없음  **abstract** **void** sendMessage(String recipient);//  // void sendMessage(String recipient){ //(4)일반 메소드  // }  } |
| //메시지 발송 클래스를 상속하는 이메일 송신 클래스  **package** p09.Polymorphism2;  **public** **class** EMailSender **extends** MessageSender {  String senderAddr;  String emailBody;  **public** EMailSender(String title, String senderName, String senderAddr,  String emailBody) {  **super**(title, senderName);  **this**.senderAddr = senderAddr;  **this**.emailBody = emailBody;  }  //Super Class의 메소드를 Override하는 메소드  **void** sendMessage(String recipient) {  System.***out***.println("---------------------------------");  System.***out***.println("제목: " + title);  System.***out***.println("보내는사람: " + senderName + " " + senderAddr);  System.***out***.println("받는 사람: " + recipient);  System.***out***.println("내용: " + emailBody);  }  } |
| //메시지 발송 클래스를 상속하는 문자메시지 송신 클래스  **package** p09.Polymorphism2;  **public** **class** SMSSender **extends** MessageSender {  String returnPhoneNo;  String message;  **public** SMSSender(String title, String senderName, String returnPhoneNo,  String message) {  **super**(title, senderName);  **this**.returnPhoneNo = returnPhoneNo;  **this**.message = message;  }  //Super Class의 메소드를 Override하는 메소드  **void** sendMessage(String recipient) {  System.***out***.println("-------------------------");  System.***out***.println("제목: " + title);  System.***out***.println("보내는 사람: " + senderName);  System.***out***.println("전화번호: " + recipient);  System.***out***.println("회신전화번호: " + returnPhoneNo);  System.***out***.println("메시지 내용: " + message);  }  } |
| **package** p09.Polymorphism2;  **public** **class** InheritanceEx08 {  **public** **static** **void** main(String[] args) {  EMailSender obj1 = **new** EMailSender("생일을 축하합니다.", "고객센터",  "admin@gmarket.co.kr", "10%할인쿠폰이 발행되었습니다.");  SMSSender obj2 = **new** SMSSender("생일을 축하 합니다.", "고객센터", "02-999-9999",  "10%할인 쿠폰이 발행 되었습니다.");  //subclass 객체(obj1,obj2)를 가지고 메소드 호출  *send*(obj1,"hatman@daum.net");  *send*(obj1,"stickman@nave.co.kr");  *send*(obj2,"010-4434-1111");  }  //슈퍼클래스 Type의 parameter변수(obj): send( )안에서는 obj가  //어느 클래스의 객체든지 상관없이 무조건 sendMessage( )를 호출한다.  **static** **void** send(MessageSender obj, String recipient){  obj.sendMessage(recipient);  }  } |
| ---------------------------------  제목: 생일을 축하합니다.  보내는사람: 고객센터 admin@gmarket.co.kr  받는 사람: hatman@daum.net  내용: 10%할인쿠폰이 발행되었습니다.  ---------------------------------  제목: 생일을 축하합니다.  보내는사람: 고객센터 admin@gmarket.co.kr  받는 사람: stickman@nave.co.kr  내용: 10%할인쿠폰이 발행되었습니다.  -------------------------  제목: 생일을 축하 합니다.  보내는 사람: 고객센터  전화번호: 010-4434-1111  회신전화번호: 02-999-9999  메시지 내용: 10%할인 쿠폰이 발행 되었습니다. |

[결론] send( )안에 있는 메소드 호출문은 obj가

EMailSender 객체이면 EmailSender 클래스의 sendMessage( )를 호출하고,

SMSSender 객체이면 SMSSender 클래스의 sendMessage( )를 호출한다. 즉, EMailsender 객체와 SMSender 객체를 MessageSender 변수에 대입해도 객체의 타입이 바뀌는 것은 아니다.

**※ 추상 메소드 선언의 두번째 목적 : 다형성**

JVM은 객체의 메소드를 호출할때 변수의 타입에 상관없이 객체가 속하는 클래스의 메소드를 호출한다.

Java Compiler는 객체가 아니라 변수의 타입만 가지고 그 메소드가 있는지 없는지 체크한다.

그렇기 때문에 MessageSender클래스에서 sendMessage( )를 없애면 컴파일 에러가 발생한다.

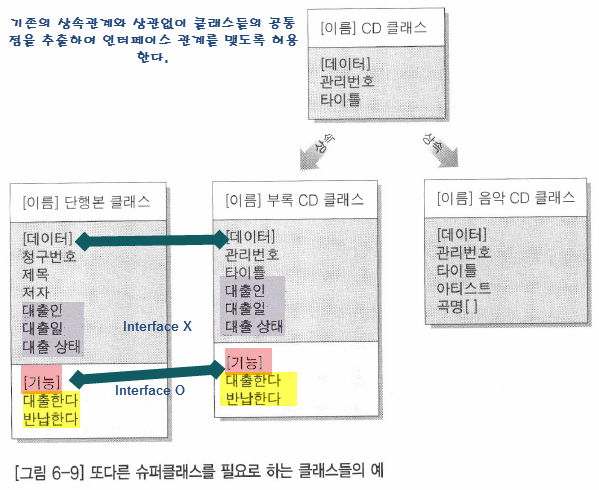
즉, 슈퍼클래스의 추상 메소드를 호출하면서 컴파일러의 체크를 무사히 통과하면서 프로그램이 실행될때는 서브클래스의 메소드가 호출되도록 하는 것이다.

**V. Interface**

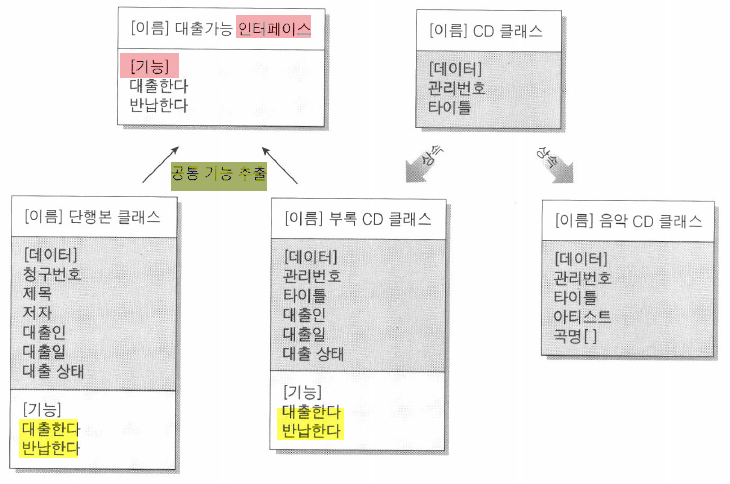
**01. 다중상속**

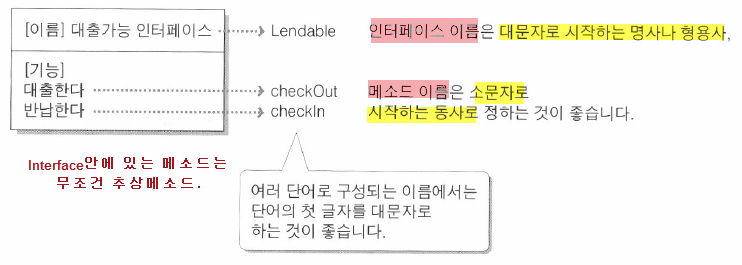
자바에서 클래스의 다중 상속을 허용하진 않지만, 때로는 이미 상속받은 서브 클래스들의 공통점을 추출해서 또다른 슈퍼클래스를 선언해서 사용하고자 할때 다중상속이 필요하다.

도서관의 자료관리 프로그램이 있을때



단행본 클래스와 부록 CD 클래스의 공통점을 추출해서 슈퍼클래스를 하나 만들어서 사용하고자 하지만, 부록클래스가 상속관계에 있으므로(자바에서는 다중상속을 허용하지 않음) 슈퍼클래스를 만들수 없다. 그래서 공통된 기능들만을 추출해서 "대출가능" 인터페이스를 만든다.





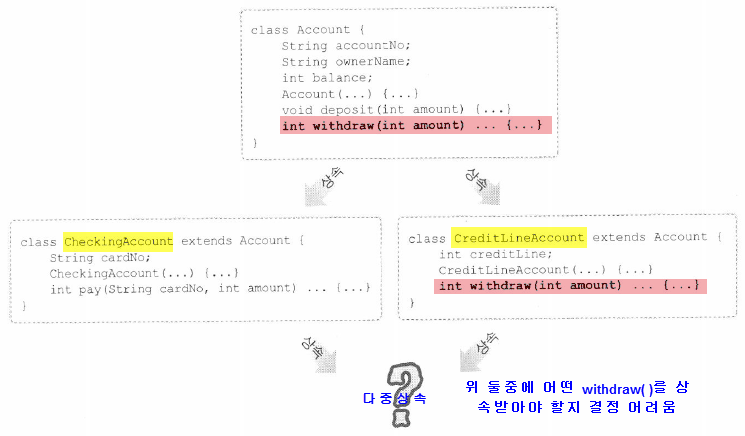
클래스와 인터페이스의 관계를 상속이라는 용어 대신에 implements(구현) 이라는 용어를 사용하고 클래

스 대신에 interface를 사용한다.

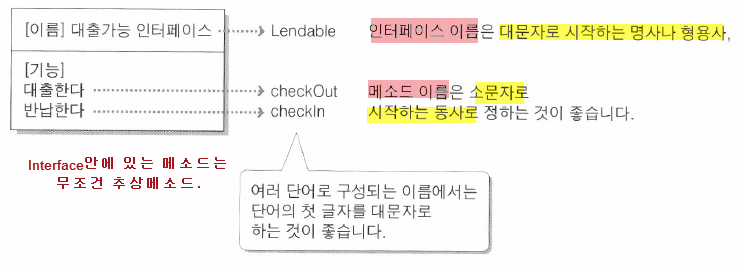
"SeparateVolume 클래스는 Lendable 인터페이스를 구현한다"

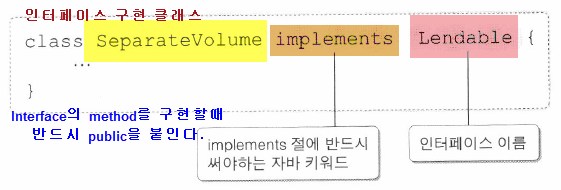
**※ 클래스의 다중상속을 허용 했을때의 문제점**

(예) 직불계좌클래스(CheckingAccount), 마이너스통장클래스(CreditLineAccount)를 동시에 상속하는 클래스를 선언할 경우

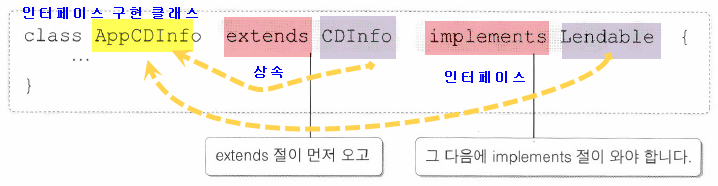


**02. Interface의 선언방법**

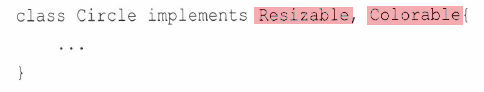




밑에 그림: 상속과 인터페이스 모두 엮여 있음



※ 하나의 클래스에 여러개의 Interface를 동시에 구현할때는 implements절에 인터페이스 이름을 기술한다.



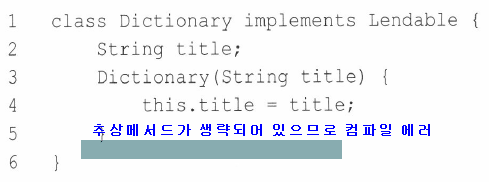
|  |
| --- |
| **package** p10.Basic.Interface;  **interface** Lendable {  // 인터페이스 메소드는 무조건 추상메소드이기 때문에 abstract를 사용하지만  // 쓰지 않아도 됨(컴파일러가 자동으로 추가)  **void** checkOut(String borrower, String date);// 대출한다  **void** checkIn(); // 반납한다  } |
| **package** p10.Basic.Interface;  **public** **class** SeperateVolume **implements** Lendable {  String requestNo;//청구번호  String bookTitle;//제목  String wirter;//저자  String borrower;//대출인  String checkOutDate;//대출일  **byte** state;//대출상태  **public** SeperateVolume(String requestNo, String bookTitle, String wirter) {  **this**.requestNo = requestNo;  **this**.bookTitle = bookTitle;  **this**.wirter = wirter;  }  //대출:: 인터페이스의 메소드를 구현시 반드시 public  **public** **void** checkOut(String borrower, String date) {  **if**(state !=0)  **return**;  **this**.borrower = borrower;  **this**.checkOutDate = date;  **this**.state = 1;  System.***out***.println("\* "+bookTitle+" 이(가) 대출되었습니다.");  System.***out***.println("대출인: "+ borrower);  System.***out***.println("대출일: "+date+"\n");  }  //반납:: 인터페이스의 메소드를 구현시 반드시 public  **public** **void** checkIn() {  **this**.borrower = **null**;  **this**.checkOutDate = **null**;  **this**.state =0;  System.***out***.println("\* "+bookTitle+" 이(가) 반납되었습니다.\n");  }  } |
| **package** p10.Basic.Interface;  **public** **class** CDInfo {  String registerNo;//관련번호  String title;//타이틀  **public** CDInfo(String registerNo, String title) {  **this**.registerNo = registerNo;  **this**.title = title;  }  } |
| **package** p10.Basic.Interface;  **public** **class** AppCDInfo **extends** CDInfo **implements** Lendable {  String borrower;  String checkOutDate;  **byte** state;  **public** AppCDInfo(String registerNo, String title) {  **super**(registerNo, title);  }  **public** **void** checkOut(String borrower, String date) {  **if** (state != 0)  **return**;  **this**.borrower = borrower;  **this**.checkOutDate = date;  **this**.state = 1;  System.***out***.println("\*" + title + "CD가 대출되었습니다.");  System.***out***.println("대출인: " + borrower);  System.***out***.println("대출일: " + date + "\n");  }  **public** **void** checkIn() {  **this**.borrower = **null**;  **this**.checkOutDate = **null**;  **this**.state = 0;  System.***out***.println("\*" + title + "CD가 반납되었습니다.\n");  }  } |
| **package** p10.Basic.Interface;  **public** **class** InterfaceEx01 {  **public** **static** **void** main(String[] args) {  SeperateVolume obj1 = **new** SeperateVolume("86738개", "개미", "베르베르");  AppCDInfo obj2 = **new** AppCDInfo("2014-9001", "Spring");  obj1.checkOut("홍길여", "20140611");  obj2.checkOut("박삼수", "20140319");  obj1.checkIn();  obj2.checkIn();  }  } |
| \* 개미 이(가) 대출되었습니다.  대출인: 홍길여  대출일: 20140611  \*SpringCD가 대출되었습니다.  대출인: 박삼수  대출일: 20140319  \* 개미 이(가) 반납되었습니다.  \*SpringCD가 반납되었습니다. |

**※ Interface의 특징**

① 객체 생성불가



② 인터페이스는 본체가 없는 추상메소드만 가질수 있으므로 인터페이스를 구현하는 서브 클래스에게 메소드의 로직을 상속할수 없지만 인터페이스를 구현하는 서브클래스의 선언방법을 제한할수 있다.



③ 인터페이스는 인터페이스의 변수 선언에도 사용된다.

**03. Interface 변수의 다형성**





|  |
| --- |
| **//단행본과 부록CD를 한꺼번에 대출**  **package** p10.Basic.Interface;  **public** **class** InterfaceEx02 {  **public** **static** **void** main(String[] args) {  Lendable arr[] = **new** Lendable[3];  arr[0] = **new** SeperateVolume("88332", "푸코의 진자", "에코");  arr[1] = **new** SeperateVolume("69933", "서양의 미술사", "곰브리치");  arr[2] = **new** AppCDInfo("02-17","Spring3.0");  *checkOutAll*(arr,"김내리", "20140709"); //배열을 파라미터로 넘겨줌  }  **static** **void** checkOutAll(Lendable arr[], String borrower, String date) {  **for** (**int** cnt = 0; cnt < arr.length; cnt++)  arr[cnt].checkOut(borrower, date);  }  } |
| \* 푸코의 진자 이(가) 대출되었습니다.  대출인: 김내리  대출일: 20140709  \* 서양의 미술사 이(가) 대출되었습니다.  대출인: 김내리  대출일: 20140709  \*Spring3.0CD가 대출되었습니다.  대출인: 김내리  대출일: 20140709 |

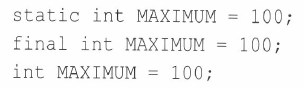
**04. Interface의 상수 필드**

인터페이스에서 인스턴스 필드는 선언할수 없지만 상수 필드는 선언 가능하다.

상수필드 선언방법은 클래스 상수필드 선언방법과 동일하다. (final static)



자바 컴파일러가 컴파일시 자동으로 final static을 추가하기 때문에 생략해도 된다.



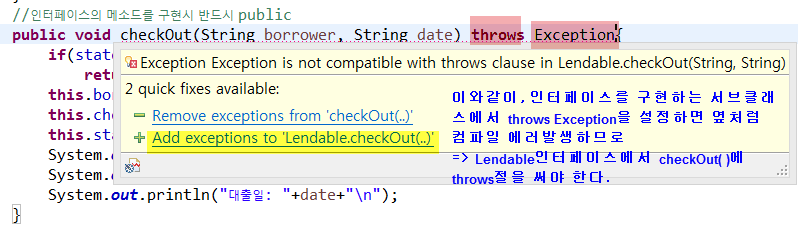
인터페이스안에 상수필드를 선언하는 것이 좋으며, 구현하는 서브클래스에 상속되기 때문에 그 클래스안에서 사용될수 있다.

|  |
| --- |
| **package** p11.constant.Interface;  **interface** Lendable {  //상수필드 (final static 생략 가능)  **final** **static** **byte** ***STATE\_BORROWED*** =1; //대출중  **final** **static** **byte** ***STATE\_NORMAL*** =0; //대출되지 않은 상태  // 인터페이스 메소드는 무조건 추상메소드이기 때문에 abstract를 사용하지만  // 쓰지 않아도 됨(컴파일러가 자동으로 추가)  **void** checkOut(String borrower, String date);// 대출한다  **void** checkIn(); // 반납한다  } |
| //대출가능 인터페이스로부터 상수 필드를 상속받는 단행본 클래스  **package** p11.constant.Interface;  **public** **class** SeperateVolume **implements** Lendable {  String requestNo;//청구번호  String bookTitle;//제목  String wirter;//저자  String borrower;//대출인  String checkOutDate;//대출일  **byte** state;//대출상태  **public** SeperateVolume(String requestNo, String bookTitle, String wirter) {  **this**.requestNo = requestNo;  **this**.bookTitle = bookTitle;  **this**.wirter = wirter;  }  //인터페이스의 메소드를 구현시 반드시 public  **public** **void** checkOut(String borrower, String date) {  **if**(state != ***STATE\_NORMAL***)//Lendable인터페스의 상수필드 사용  **return**;  **this**.borrower = borrower;  **this**.checkOutDate = date;  **this**.state = ***STATE\_BORROWED***;//Lendable인터페이스의 상수필드 사용  System.***out***.println("\* "+bookTitle+" 이(가) 대출되었습니다.");  System.***out***.println("대출인: "+ borrower);  System.***out***.println("대출일: "+date+"\n");  }  //인터페이스의 메소드를 구현시 반드시 public  **public** **void** checkIn() {  **this**.borrower = **null**;  **this**.checkOutDate = **null**;  **this**.state = ***STATE\_NORMAL***;//Lendable인터페이스의 상수필드 사용  System.***out***.println("\* "+bookTitle+" 이(가) 반납되었습니다.\n");  }  } |
| **//Lendable인터페이스를 구현하지 않았을경우 상수필드 사용시 인터페이스 이름과 마침표 사용**  **package** p11.constant.Interface;  **public** **class** InterfaceEx03 {  **public** **static** **void** main(String[] args) {  SeperateVolume obj = **new** SeperateVolume("86444", "나무", "베로베로");  *printState*(obj);  obj.checkOut("김미정", "20140311");  *printState*(obj);  }  **static** **void** printState(SeperateVolume obj) {  //Lendabel인페이스를 구현하지 않은 클래스이므로 필드이름앞에 인터페이스이름과 마침표 한다.  **if** (obj.state == Lendable.***STATE\_NORMAL***) {  System.***out***.println("-----------------------");  System.***out***.println("대출상태 : 대출가능");  System.***out***.println("-----------------------");  }  **if** (obj.state == Lendable.***STATE\_BORROWED***) {  System.***out***.println("-----------------------");  System.***out***.println("대출상태 : 대출중");  System.***out***.println("대출인: " + obj.borrower);  System.***out***.println("대출인: " + obj.checkOutDate);  System.***out***.println("-----------------------");  }  }  } |
| -----------------------  대출상태 : 대출가능  -----------------------  \* 나무 이(가) 대출되었습니다.  대출인: 김미정  대출일: 20140311  -----------------------  대출상태 : 대출중  대출인: 김미정  대출인: 20140311  ----------------------- |

**04. Interface에서 Exception을 발생하는 Abstract Method**

일반적으로 Exception이 빌생 했을때 Method밖으로 던지기 위해서 throws절을 사용한다. 추상 메서드를 구현하는 메소드의 경우에는 throws절을 써도 메소드 밖으로 익셉션을 던질수 없을때가 있다.

그럴때는 Lendable 인터페이스에서 checkOut( )에 throws절을 쓴다.



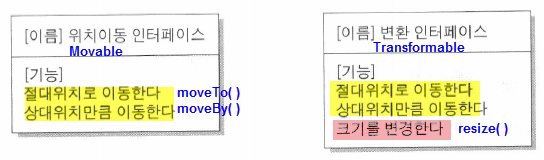
SeperateVolume 클래스에 throws Exception만 쓰면 컴파일 에러 발생, Lendable에 반드시 throws Exception

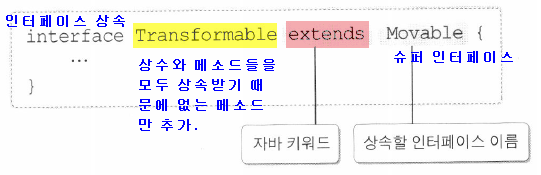
|  |
| --- |
| **package** p12.Exception.Interface;  **interface** Lendable {  //상수필드 (final static 생략 가능)  **final** **static** **byte** ***STATE\_BORROWED*** =1; //대출중  **final** **static** **byte** ***STATE\_NORMAL*** =0; //대출되지 않은 상태  // 인터페이스 메소드는 무조건 추상메소드이기 때문에 abstract를 사용하지만  // 쓰지 않아도 됨(컴파일러가 자동으로 추가)  abstract **void** checkOut(String borrower, String date) **throws** Exception;// 대출한다  abstract **void** checkIn(); // 반납한다  } |
| **package** p12.Exception.Interface;  **public** **class** SeperateVolume **implements** Lendable {  String requestNo;//청구번호  String bookTitle;//제목  String wirter;//저자  String borrower;//대출인  String checkOutDate;//대출일  **byte** state;//대출상태  **public** SeperateVolume(String requestNo, String bookTitle, String wirter) {  **this**.requestNo = requestNo;  **this**.bookTitle = bookTitle;  **this**.wirter = wirter;  }  //인터페이스의 메소드를 구현시 반드시 public  **public** **void** checkOut(String borrower, String date) **throws** Exception{  //대출중 상태(1)이면 메소드 밖으로 익셥션을 던진다  **if**(state !=0)  // return;  **throw** **new** Exception("대출불가: " + bookTitle);  **this**.borrower = borrower;  **this**.checkOutDate = date;  **this**.state = 1;  System.***out***.println("\* "+bookTitle+" 이(가) 대출되었습니다.");  System.***out***.println("대출인: "+ borrower);  System.***out***.println("대출일: "+date+"\n");  }  //인터페이스의 메소드를 구현시 반드시 public  **public** **void** checkIn() {  **this**.borrower = **null**;  **this**.checkOutDate = **null**;  **this**.state =0;  System.***out***.println("\* "+bookTitle+" 이(가) 반납되었습니다.\n");  }  } |
|  |

**05. Interface 상속**

(1) 인터페이스 상속 방법

(예) 그래프를 그리는 프로그램 작성시

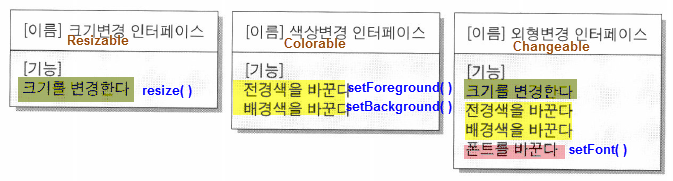


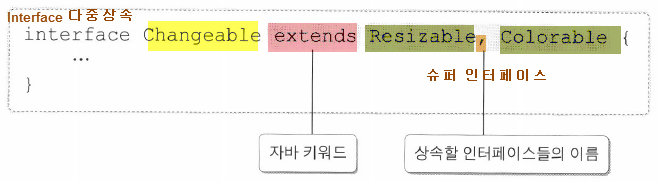


|  |
| --- |
| //위치이동 인터페이스 선언  **package** p13.Inherit.Interface;  **public** **interface** Movable {  **void** moveTo(**int** x, **int** y);// 절대위치로 이동한다.  **void** moveBy(**int** xOffset, **int** yOffset); // 상대위치만큼 이동한다.  } |
| //변환 인터페이스 선언  **package** p13.Inherit.Interface;  **public** **interface** Transformable **extends** Movable {  **void** resize(**int** width, **int** height); // 크기를 변경한다.  } |
| //변환 인터페이스를 구현하는 사각형 클래스  **package** p13.Inherit.Interface;  **public** **class** Rectangle **implements** Transformable {  **int** x, y, width, height;  **public** Rectangle(**int** x, **int** y, **int** width, **int** height) {  **this**.x = x;  **this**.y = y;  **this**.width = width;  **this**.height = height;  }  **public** **void** moveTo(**int** x, **int** y) {//Movable 인터페이스의 메소드를 구현  **this**.x = x;  **this**.y = y;  }  **public** **void** moveBy(**int** xOffset, **int** yOffset) {  **this**.x += xOffset;  **this**.y += yOffset;  }  **public** **void** resize(**int** width, **int** height) {//Transformabable 인터페이스의 메소드 구현  **this**.width = width;  **this**.height = height;  }  } |
| //Rectangle클래스를 사용하는 프로그램  **package** p13.Inherit.Interface;  **public** **class** InterfaceEx04 {  **public** **static** **void** main(String[] args) {  Rectangle obj = **new** Rectangle(100, 200, 10, 10);  *printRectangle*(obj);  obj.moveTo(25, 35);  *printRectangle*(obj);  obj.moveBy(-5, -5);  *printRectangle*(obj);  obj.resize(30, 30);  *printRectangle*(obj);  }  **static** **void** printRectangle(Rectangle obj) {  System.***out***.printf("사각형: 위치(%d, %d) 크기(%d x %d) %n", obj.x, obj.y,  obj.width, obj.height);  }  } |
| 사각형: 위치(100, 200) 크기(10 x 10)  사각형: 위치(25, 35) 크기(10 x 10)  사각형: 위치(20, 30) 크기(10 x 10)  사각형: 위치(20, 30) 크기(30 x 30) |

**(2) Interface의 다중상속**

(예)





|  |
| --- |
| //크기변경 인터페이스  **package** p14.Inherit2.Interface;  **public** **interface** Resizable {  **void** resize(**int** width, **int** height);  } |
| //외형변경 인터페이스  **package** p14.Inherit2.Interface;  **public** **interface** Changeable **extends** Resizable, Colorable {  **void** setFont(String font);  } |
| //외형변경 인터페이스  **package** p14.Inherit2.Interface;  **public** **interface** Changeable **extends** Resizable, Colorable {  **void** setFont(String font);  } |
| **//외형변경 인터페이스를 구현하는 라벨 클래스**  **package** p14.Inherit2.Interface;  **public** **class** Label **implements** Changeable {  String text;  **int** width, height;  String foreground, background;  String font;  **public** Label(String text, **int** width, **int** height, String foreground,  String background, String font) {  **this**.text = text;  **this**.width = width;  **this**.height = height;  **this**.foreground = foreground;  **this**.background = background;  **this**.font = font;  }  **public** **void** resize(**int** width, **int** height) {  **this**.width = width;  **this**.height = height;  }  **public** **void** setForeground(String color) {  **this**.foreground = color;  }  **public** **void** setBackground(String color) {  **this**.background = color;  }  **public** **void** setFont(String font) {  **this**.font = font;  }  } |
| **package** p14.Inherit2.Interface;  **public** **class** InterfaceEx05 {  **public** **static** **void** main(String[] args) {  Label obj = **new** Label("안녕하세요", 100, 30, "yellow", "green", "굴림체");  *printLabel*(obj);  obj.resize(200, 70);  obj.setFont("black");  obj.setBackground("white");  obj.setFont("궁서체");  *printLabel*(obj);  }  **static** **void** printLabel(Label obj) {  System.***out***.printf("%s: %d x %d 배경색(%s) 전경색(%s) 폰트(%s) %n", obj.text,  obj.width, obj.height, obj.background, obj.foreground, obj.font);  }  } |
| 안녕하세요: 100 x 30 배경색(green) 전경색(yellow) 폰트(굴림체)  안녕하세요: 200 x 70 배경색(white) 전경색(yellow) 폰트(궁서체) |