5.1 상속의 개념

상속: 부모 클래스에서 만든 필드,메소드를 자식 클래스가 물려 받는 것.

자식클래스가 부모 클래스의 필드, 메소드를 만들지 않고도 만든 것과 같은 효과 가짐

상속은 클래스 사이의 상속! 객체 상속 no!

1. 상속의 필요성 : 클래스의 간결화( 멤버의 중복 작성 불필요), 클래스 관리 용이, 소프트웨어의 생산성 확장(클래스 재사용과 확장 용이)

Ex) 부모클래스인 Person 이 말하기, 먹기, 움직이기 속성을 가지고 있으면 이를 상속받은 자식 클래스인 Student 는 부모의 속성 + 자신의 고유 속성인 공부하기 도 갖게 됨.

5.2 클래스 상속과 객체

1) 자바의 상속 선언:

부모클래스(super class), 자식 클래스(sub class) 라 부름

Ex) public class Student(자식) extends Person(부모)

1. 상속과 객체

Person p = new Person() //부모클래스 객체생성-부모클래스의 멤버만 가짐

Student s = new Student(); //자식클래스 객체생성 – 부모클래스와 자식 클래스 멤버 모두 가짐. 부모클래스의 private 멤버 제외하곤 접근 가능!

1. 자바 상속의 특징

* 클래스는 하나의 부모만 상속함! Extends 뒤에는 하나의 부모만 존재!
* 상속 횟수에 제한 x
* 자바 계층 구조의 최상위는 java.lang.Object

5.3 상속과 protected 접근 지정자

1) 슈퍼 클래스에 대한 접근 지정

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 슈퍼클래스 멤버에 접근하는 클래스 종류 | 슈퍼 클래스 멤버의 접근 지정자 | | | |
| private | default | protected | Public |
| 같은 패키지 클래스 | X | O | O | O |
| 다른 패키지 클래스 | X |  |  | O |
| 같은 패키지 서브클래스 | X | O | O | O |
| 다른 패키지 서브클래스 | X |  | O | O |

슈퍼클래스의 private 멤버: 현재 클래스 멤버만 접근 가능

슈퍼클래스의 default 멤버: 같은 패키지 내 모든 클래스 접근 가능

슈퍼클래스의 protected멤버: 같은 패지키 내 클래스 혹은 상속받은 서브 클래스

슈퍼클래스의 public 멤버: 모든 클래스 접근 가능

5.4 상속과 생성자

1) 서브 클래스와 슈퍼클래스의 생성자 호출 및 실행

* 서브 클래스 객체가 생성될 때!! 서브클래스 생성자는 필요한 초기화 수행하고 슈퍼클래스 생성자는 생성된 객체 속에 있는 슈퍼 클래스 멤퍼 초기화를 각각 수행함.
* 서브 클래스 객체의 생성자가 먼저 호출 되지만 슈퍼클래스의 생성자가 먼저 실행된 후 서브 클래스 생성자 실행됨.

2) 서브 클래스에서 슈퍼 클래스 생성자 선택

* 슈퍼 클래스의 기본 생성자가 자동 선택되는 경우

개발자의 명시적 지시가(super를 사용하지 않으면) 없으면 컴파일러는 자동으로 슈퍼클래스의 기본 생성자(매개변수를 받지 않는 생성자)를 호출

* Super() 를 이용하여 명시적으로 슈퍼 클래스의 생성자 선택

생성자의 첫 라인에 사용해야 함

Super() 슈퍼 클래스 생성자를 호출하는 코드

5.5 업캐스팅과 instanceof 연산자

캐스팅: 타입변환

캐스팅은 1.다운캐스팅과 2.업캐스팅으로 나뉨.

1. 업캐스팅: 서브 클래스의 객체에 대한 레퍼런스를 슈퍼 클래스 타입으로 변환하는 것.

슈퍼클래스의 레퍼런스로 서브 클래스의 객체를 가리킴.

Ex) Person p;

Student s = new Student();

p = ; // p=(Person) s; 업캐스팅!(슈퍼클래스 레퍼런스p가 서브 클래스 객체에 대한 레퍼런스 s를 가리킴. 하지만 p는 s의 Person 클래스의 멤버만 접근 가능.

1. 다운 캐스팅:

Ex) Person p = new Student(); -업 캐스팅

Student s;

s = (Student) p ; //다운 캐스팅 (슈퍼클래스 레퍼런스 p의 타입을 서브클래스인 Student 타입으로 변환함으로써 Student 클래스의 멤버도 접근가능

1. 업캐스팅과 instanceof 연산자

업캐스팅 된 레퍼런스 p가 어떤 객체를 가리키는 것인지 모름.

Ex) Person p = new Person();

Person p = new Student();

Person p = new Researcher();

그래서 instanceof 연산자 이용.

Ex) if(kim instanceof Person) //kim 이 Person 타입인지?

5.6 메소드 오버라이딩

1) 메소드 오버라이딩 : 슈퍼클래스-서브클래스 메소드 사이에 발생.

슈퍼클래스-서브클래스에 선언된 메소드와 같은 이름, 같은 리턴타입, 같은 매개변수 리스트를 가짐. 동일한 이름의 메소드를 서브 클래스에 재작성!

동적바인딩: 슈퍼 클래스 메소드를 무시하고 서브 클래스에서 오버라이딩된 메소드가 무조건 실행되도록 하는 것.

Ex1) Line line = new Line();

line.draw(); // Line클래스의 draw()호출

ex2) Shape shape = new Line(); // 업캐스팅

shape.draw(); //Line의 draw() 호출

2) 오버라이딩의 목적, 다형성 실현

목적: 슈퍼 클래스에 선언된 메소드를 각 서브 클래스들이 자신만의 내용으로 새로 구현하는 것.

1. 오버라이딩의 제약 사항

* 슈퍼클래스 메소드와 동일한 형태 (동일한 메소드 이름, 매개변수 타입, 개수, 리턴타입)
* 슈퍼클래스 메소드의 접근지정자(public>protected>default>private)보다 접근 범위를 좁혀서 오버라이딩 X
* Static, private, final 로 선언된 메소드는 서브 클래스에서 오버라이딩 할 수 없음.

1. 메소드 오버라이딩 활용
2. 동적 바인딩: 오버라이딩된 메소드 호출

오버라이딩이 되어 있는 경우 동적바인딩에 의해 항상 서브 클래스에 오버라이딩한 메소드가 호출 .

1. 오버라이딩과 super 키워드

Super.을 이용하면 슈퍼 클래스의 필드와 메소드 접근 가능.

1. 오버로딩과 오버라이딩

|  |  |  |
| --- | --- | --- |
|  | 메소드 오버로딩 | 메소드 오버라이딩 |
| 선언 | 슈퍼클래스-서브클래스 or  같은 클래스 내  동일한 이름의 메소드 중복 작성 | 슈퍼클래스-서브클래스  서브클래스에서 슈퍼클래스의 메소드와 동일한 이름(같은 리턴타입, 같은 매개변수)으로 메소드 재작성 |
| 관계 | 1.상속관계  2.동일한 클래스 내 | 상속관계 |
| 목적 | 이름이 같은 여러 개의 메소드를 작성 | 슈퍼클래스 메소드 무시하고 서브클래스에서 새로운 기능의 메소드를 재정의 |
| 조건 | 메소드 이름 동일  매개변수 타입, 개수 다름 | 메소드 이름,  매개변수 타입, 개수, 리턴 타입 동일 |
| 바인딩 | 정적 바인딩.  호출될 메소드는 컴파일 시에 결정 | 동적바인딩,  실행 시간에 오버라이딩된 메소드 찾아 호출 |

5.7 추상 클래스

1) 추상 메소드 : 선언만 되어 있고 내용은 없는 메소드. Abstract 키워드가 필요 !

Ex) public abstract String getName();

2) 추상 클래스 만들기 : abstract 키워드로 선언해야함

* 추상 메서드를 포함하는 클래스 -> 추상클래스로 선언되어야 함

Ex) abstract class Shape //추상 클래스

public Shape() { }

abstract public void draw(); //추상 메서드

* 추상 메소드가 없지만 abstract 로 선언한 클래스

Ex\_) abstract class Shape{ //추상 클래스

String name;

public void draw( String name){ }

1. 추상 클래스는 객체 생성할 수 없다
2. 추상 클래스의 상속

추상클래스를 상속받는 서브클래스는 추상클래스가 됨. 서브클래스에 abstract 붙여야 함

1. 추상 클래스의 구현과 목적

슈퍼 클래스에 선언된 모든 추상 메소드를 서브 클래스에서 오버라이딩하여 실행 가능한 코드로 구현하는 것.

추상 슈퍼 클래스는 인터페이스 역할을 하고 서브 클래스는 추상 메소드를 목적에 맞게 구현하는 다형성을 실현함!

1. 추상 클래스의 용도

서브 클래스에서 추상 클래스에 선언된 추상 메소드를 모두 구현해야함!!!

5.8 인터페이스

1) 자바의 인터페이스

* Interface 키워드 사용해 선언,
* 구현되지 않은 추상 메소드를 가질 수 있어 객체 생성 할 수 없음
* 인터페이스 타입의 레퍼런스 변수 선언 가능함
* 인터페이스끼리 상속 가능
* 인터페이스 상속받아 클래스 작성하면 인터페이스의 모든 추상 메소드를 구현해야 함

Ex) interface PhoneInterface{

Public static final int Timeout = 10000;

Public abstract void sendCall();

Public abstract void receiveCall();

}

2) 인터페이스 구현

Implements 키워드 사용

Ex) class SansungPhone implements PhoneInterface{ // 인터페이스 폰인터페이스를 구현하는 클래스 삼성폰

}

2) 인터페이스 상속

클래스는 인터페이스를 상속받을 수 없고 인터페이스끼리만 상속 가능

Ex) interface MobilePhoneInterface extends PhoneInterface //두개의 인터페이스

인터페이스 다중 상속 가능

Ex) interface MusicPhoneInterface extends MobilePhoneInterface, Mp3Interface{

}

1. 인터페이스 목적

인터페이스는 클래스들이 기능을 서로 다르게 구현할 수 있도록 하는 클래스의 규격 선언이며 클래스의 다형성을 실현하는 도구

1. 다중 인터페이스 구현

클래스는 하나 이상의 인터페이스 구현 할 수 있다. 각 인터페이스에 선언된 모든 추상 메소드를 구현해야함.

1. 클래스 상속과 함께 인터페이스 구현

클래스 상속 받으면서 동시에 인터페이스 구현 가능.

Ex) class SmartPhone extends PDA implements MobilePhoneInterface, MP3Interface{

}

6) 인터페이스와 추상 클래스 비교

|  |  |  |
| --- | --- | --- |
|  | 추상클래스 | 인터페이스 |
| 목적 | 추상 클래스는 서브 클래스에서 필요로 하는 대부분의 기능을 구현해 두고, 서브 클래스가 상속받아 활용할 수 있도록 함.  서브 클래스에서 구현해야 하는 기능만을 추상 메소드로 선언하여 서브 클래스에서 구현하도록 하는 목적 | 인터페이스를 상속받는 클래스의 목적에 따라 인터페이스의 모든 추상 메소드를 만들도록 하는 목적 |
| 구성 | 추상메소드, 일반메소드 모두 포함  상수, 변수 필드 모두 포함 | 변수필드는 포함하지 않음  상수, 추상메소드, 일반 메소드, 디폴트 메소드, static메소드 모두 포함,  Protected 접근 지정 선언 불가  다중 상속 지원 |