상속과 인터페이스

학습 목표

- 상속의 개념과 상속 관련 키워드를 학습합니다.
- 상속 후에 멤버 변수를 사용하는 방법에 대하여 학습합니다.
- 상속 후에 메소드 overriding 하는 방법과 상속 관련 modifier를 학습합니다.
- 상속과 생성자를 자동 호출하는 super 키워드에 대하여 학습합니다.
- 객체 간 형변환하는 방법과 형변환을 통한 다형성을 학습합니다.
- 인터페이스의 개념과 인터페이스를 구현하는 방법을 학습합니다.

상속과 인터페이스

- 상속이란
- 멤버 변수의 상속
- 메소드의 상속
- 생성자 호출과 super 키워드
- 객체의 형변환과 다형성
- 인터페이스란
- 인터페이스의 활용

상속이란

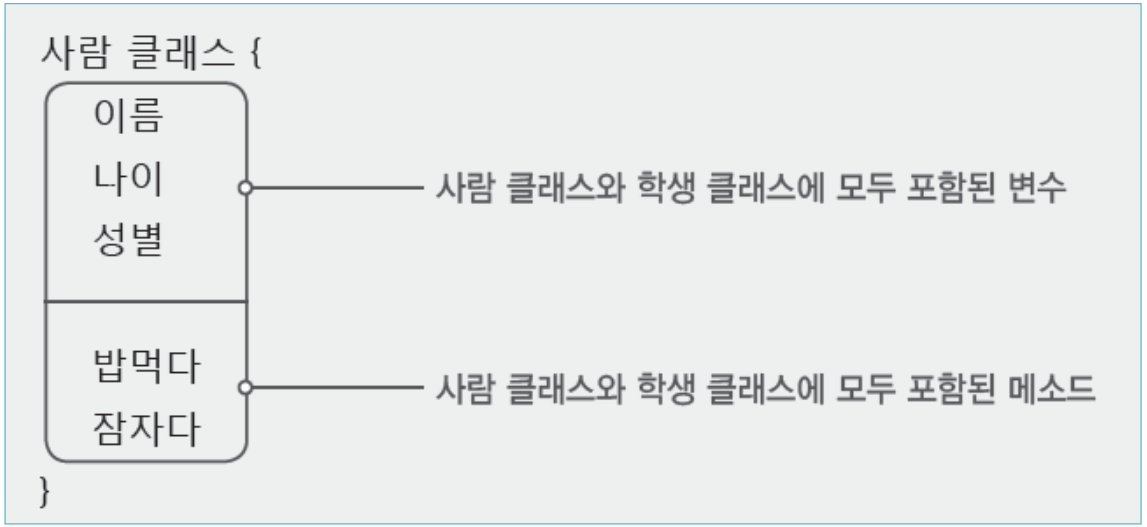
- 상속의 개념과 코드 재사용
- 상속과 extends 키워드

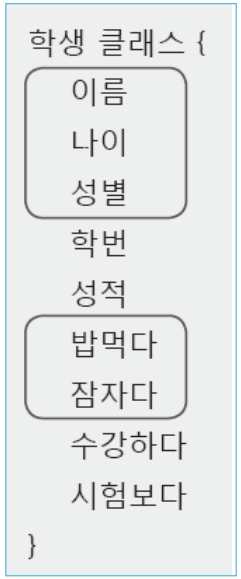
상속의 개념과 코드 재사용

* 일반화와 특별화의 관계
* ~이다 관계(is-a relationship)

학생은 사람이다
Student is a Person

* 사람 클래스 학생 클래스





상속의 개념과 코드 재사용

* 사람 클래스(상위클래스)와 학생 클래스(하위클래스)의 상속

```
사람 클래스 {
    이름
    나이
    성별
    밥먹다
    잠자다
}
사람 클래스를 상속받은 학생 클래스 {
    학번
    성적
    수강하다
    시험보다

}
```

학생 클래스에 추가할 변수와 메소드만 정의

상속과 extends 키워드

- 상속 표현 키워드
- 클래스 선언부에서 사용
- extends 키워드 뒤에는 하나의 클래스만 상위 클래스로 지정 가능:단일 상속

```
[modifiers] class 클래스이름
[extends 상위클래스이름] ○──────────── 상위 클래스 상속
[implements 인터페이스이름1, 인터페이스이름2, ...] {
    // 클래스 멤버 부분
}
```

상속과 extends 키워드

* Person 클래스와 Student 클래스를 상속받아 정의

```
class Person {
    String name;        /* 이름 */
    int age;            /* 나이 */
}
```

```
class Student  extends Person  {
    String name;            /* 이름 */
    int number;             /* 학번 */
    int age;                /* 나이 */
    int grade;              /* 학년 */
    String schoolName;      /* 학교명 */
    String records;         /* 성적 */
}
```

상속과 extends 키워드

- extends 가 명시되지 않은 클래스는 java.lang.Object 클래스를 자동 상속

```
class Employee {
    //...
}
```

동일한 의미

```
class Person extends Object {
    //...
}
```

- 클래스간의 다중상속 불가능

```
class Graduate extends Student, Professor {     /*다중 상속 불가능. 오류 발생*/
    //...
}
```

멤버 변수의 상속

- 상속받은 멤버 변수의 사용
- 상속과 private 멤버 변수
- 동일한 이름을 가진 멤버 변수의 사용

상속받은 멤버 변수의 사용

- 상위클래스의 멤버 변수 자동 포함
- 프로그램 6-1 실습
  - StudentTest.java 작성
  - Person 클래스를 상속받아서 Student 클래스 정의
  - 실행 결과

```
C:₩JAVA>Java StudentTest
홍길동 학생은 학번은 0820543 이고 현재 1 이며 이번 성적은 A 입니다.
```

상속과 private 멤버 변수

- private 변수는 하위클래스에서 접근 불가

```
class Person {
    ......
    private String number = "890608-1234567"; ◦——— Person의 private 변수
    ......
}

class Student extends Person { ◦————————————— Person 상속
    ......
    void testPrivate() {
        System.out.prinltn(number); ◦————————— Person의 private 변수 사용.
    }                                          오류 발생
    ......
}
```

동일한 이름을 가진 멤버 변수의 사용

- 상위클래스와 하위클래스에 동일한 이름의 변수가 존재 가능
- 하위클래스에 선언된 변수 우선 사용

```
class Person {
    ......
    String number = "890608-1234567";   ← Person의 number는 주민번호를 나타내는 변수
    ......
}

class Student extends Person {   ← Person 상속. number 변수도 상속받아 자동 포함.
    ......
    String number = "0820423";   ← Student의 number는 학번을 나타내는 변수.
                                   Student 내에는 number 변수가 두 개 포함된 상태.
    void testPrivate() {
        System.out.prinltn(number);   ← Student에 선언된 number 변수가 우선적으로 사용.
                                        "0820423" 출력
    }
    ......
}
```

동일한 이름을 가진 멤버 변수의 사용

- 프로그램 6-2 실습
  - StudentMemberTest.java 작성
  - 하위클래스에서 상위클래스의 변수 사용
  - 코멘트 해제 결과

```
C:₩JAVA>Javac StudentMemberTest.java
StudentMemberTest.java:14: age has private access in Person
        System.out.println(age);
                           ^
1 error
```

  - 코멘트 유지 결과

```
C:₩JAVA>java StudentMemberTest
홍길동 학생은 학번은 0820534 이고 현재 1 이며 이번 성적은 A 입니다.
```

메소드의 상속

- 메소드 overriding
- final modifier
- abstract modifier

메소드 overriding

* 상위클래스 메소드는 그대로 하위클래스에서 사용 가능

```
class Person {
    ......
    void print() {
        System.out.println
        (" 이 사람의 이름은 " + name
        + " 이고 나이는 " + age + " 입니다.");
}

class Student  extends Person  {
    ......
}

class StudentTest {
    public static void main(String args[]){
        Student stu = new Student();
        stu.print(); ○———————— 상속받은 Person 클래스의 print() 메소드 실행
    }
```

메소드 overriding

* 상속 후에 하위클래스에서 동일 이름의 메소드 재정의 가능:메소드 overriding

```
class Person {
    ......
    void print() {
        System.out.println
        (" 이 사람의 이름은 " + name
        + " 이고 나이는 " + age + " 입니다.");
    }
```

```
class Student  extends Person  {
    ......
    void print(){
        System.out.println
        (" 이 사람의 이름은 " + name
        + " 이고 나이는 " + age + " 입니다.");
        System.out.println
        (name + " 은 학번이 " + number
        + " 이고 이번 성적은 " + records + " 입니다.");
    }
```

```
class StudentTest {
    public static void main(String args[]){
        Student stu = new Student();
        stu.print(); ———— Student 클래스에서 overriding된 print() 메소드 실행
    }
}
```

메소드 overriding

- 메소드 overriding 규칙
  - 메소드 이름, 리턴 타입, 매개변수의 선언은 반드시 같아야 합니다.
  - 접근 modifier는 하위 클래스의 접근 범위가 넓어지는 쪽으로는 변경이 가능합니다.
  - 더 많은 종류의 예외(exception)를 throws로 선언하여 던질 수는 없습니다. 따라서 상위 클래스의 메소드에서 throws한 예외의 상위 클래스 예외를 throws할 수 없습니다.

메소드 overriding

* 리턴 타입 오류

```
class MyBase {
    public void aMethod(int num1) {
        // ...
    }
}
class MyDerived extends MyBase {
    public int aMethod(int num1) {
        // ......
        return num1;
    }
}
```

num1 변수 값 리턴하므로 리턴 타입은 int

매개변수 오류

```
class MyBase {
    public void aMethod(int num1) {
        // ...
    }
}
class MyDerived extends MyBase {
    public void aMethod(String num1) {
        // ...
    }
}
```

메소드 overriding

* modifier 오류

```
class MyBase
    public void aMethod(int num1) {
class MyDerived extends MyBase {
    void aMethod(int num1) {
        // ...
    }
}
```

예외 지정 오류

```
class MyBase {
    void aMethod(int num1) throws IOException {
        // ...
    }
}
class MyDerived extends MyBase {
    public void aMethod(int num1) throws Exception {
        // ...
    }
}
```

메소드 overriding

- 프로그램 6-3 실습
  - StudentMethodTest.java 작성
  - non-overriding 메소드 이용
  - 실행 결과

```
C:₩JAVA>java StudentMethodTest
이 사람의 이름은 홍길동 이고 나이는 20 입니다.
```

메소드 overriding

- 프로그램 6-4 실습
  - OverridingTest.java 작성
  - 프로그램 6-3을 수정하여 overriding 메소드 이용
  - 실행 결과

```
C:₩JAVA>java OverridingTest
이 사람의 이름은 홍길동 이고 나이는 20 입니다.
홍길동 은 학번이 0820543 이고 이번 성적은 A 입니다.
```

final modifier

- 클래스, 변수, 메소드 선언에 사용
- final 변수는 상수

```
final int  MAX_PRIORITY = 20;
.....
MAX_PRIORITY = MAX_PRIORITY++; ←———— final 변수는 값 수정 불가능. 오류 발생
```

- final 클래스는 상속 불가능한 클래스 선언

```
final class Account {
    // .....
}
class CheckAccount extends Account {  ←———— Account 클래스는 final이므로
                                              상속 불가능. 오류 발생.
    // .....
}
```

final modifier

- final 메소드는 overriding 불가능한 메소드 선언

```
class Account {
    String passwd;
    String name;
    final void setPasswd(String  newPasswd) {
        passwd = newPasswd;
    }
}

class CheckAccount extends Account {
    void setPasswd(String newPwd) {
        ;
    }
}
```

final 메소드 setPasswd는 Account 클래스의 하위 클래스에서 메소드 overriding 불가능

Account 클래스의 setPasswd 메소드는 final로 선언되었으므로 메소드 overriding 불가능. 오류 발생

abstract modifier

- 클래스와 메소드 선언에 사용
- abstract 클래스는 객체 생성 불가능한 클래스로 상속만 가능

```
abstract class GeoShape
```

```
GeoShape oneShape = new GeoShape();  ←——— 오류 발생
```

- abstract 메소드를 하나 이상 포함하면 abstract 클래스로 선언해야 함
- 메소드 선언부만 있고 구현부는 없는 메소드는 abstract 메소드로 선언해야 함

abstract modifier

- abstract 클래스의 하위클래스는 abstract 메소드 overriding 필요

```
abstract class GeoShape {
    int x;
    int y;
    abstract void drawShape(Graphics g);
}

class Rectangle extends GeoShape {
    int width;
    int height;
    void drawSahpe(Graphics g) {
        g.drawRect( x, y, width, height );
    }
}
```

abstract 메소드 drawShape를 포함하는 abstract 클래스 GeoShape 정의

abstract 클래스 GeoShape를 상속받아 Rectangle 클래스 정의. GeoShape 클래스의 객체 생성은 불가능

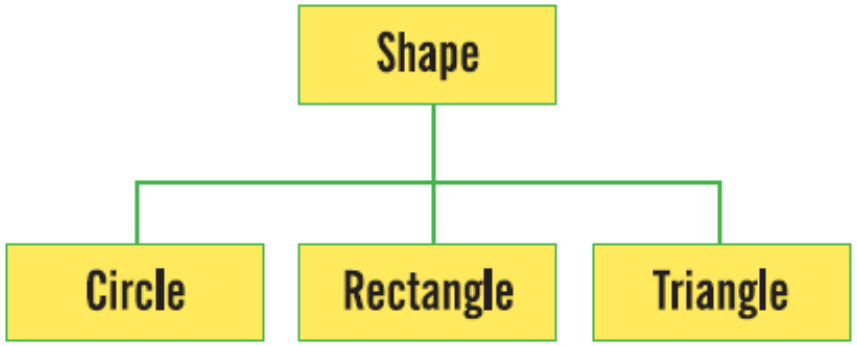
astract 클래스 GeoShape를 상속받아서 drawRect 메소드 overriding. 만약 overriding 하지 않는다면 오류 발생

abstract modifier

- abstract 클래스로 정의하는 경우
  - 클래스가 한 개 이상의 abstract 메소드를 갖고 있을 때
  - abstract 클래스로부터 파생된 클래스가 상위 클래스에 있는 abstract 메소드를 오버라이딩(overriding) 하여 구현하지 않았을 때
  - 클래스가 인터페이스로부터 파생되었는데 인터페이스에 있는 전체 메소드를 구현하지 않았을 때
- final과 abstract 동시 사용 불가능

abstract modifier

- 프로그램 6-5 실습
  - AbstractTest.java 작성
  - abstract 클래스 Shape 상속
  -



실행 결과

```
C:₩JAVA>java AbstractTest
r1의 너비 = 20.0
r1의 둘레 = 18.0
c1의 너비 = 78.53981633974483
c1의 둘레 = 31.41592653589793
```

생성자 호출과 super 키워드

- 생성자 자동 호출
- super 키워드와 상위 클래스의 멤버 사용
- super 키워드와 상위 클래스의 생성자 호출

생성자 자동 호출

- 하위클래스 객체 생성시에 상위클래스 객체도 같이 생성
- 하위클래스 생성자 호출시에 자동으로 상위클래스 생성자 자동 호출 가능
- 하위클래스 생성자 실행 이전에 상위클래스 생성자 먼저 실행

생성자 자동 호출

- 프로그램 6-6 실습
  - ConstructorTest.java 작성
  - 상위클래스 기본 생성자 자동 호출
  - 실행 결과

```
C:₩JAVA>java ConstructorTest
클래스 Super의 생성자 수행
클래스 Sub의 생성자 수행
100.0
1000.0
```

super 키워드와 상위 클래스의 멤버 사용

- 상위클래스 객체 참조 키워드
- 상위클래스와 하위클래스에 동일 이름의 변수나 메소드 구분
- static 메소드 내부에서 사용 불가능한 키워드

```
class MyBase {
    int memVar = 100;
    int aMethod(int    num1) {
        return num1 * 100;
    }
}
```

```
class MyDerived extends MyBase {
    int memVar = 200;
    public void aMethod(int    memVar) {
        this.memVar = memVar;          // 라인1.
        super.aMethod(this.memVar);    // 라인2.
        System.out.println(super.memVar); // 라인3.
    }
}
```

super 키워드와 상위 클래스의 멤버 사용

- 프로그램 6-7 실습
  - SuperMemberTest.java 작성
  - super 키워드를 이용하여 상위클래스의 멤버 변수 구분
  - 실행 결과

```
C:₩JAVA>java SuperMemberVarTest
이 사람의 이름은 홍길동 이고 나이는 20 입니다.
홍길동 은 주민등록번호가 890608-1234567 이고
학번이 0820543 이며 이번 성적은 A 입니다.
```

super 키워드와 상위 클래스의 멤버 사용

- 프로그램 6-8 실습
  - SuperMethodTest.java 작성
  - super 키워드를 이용하여 상위클래스의 메소드 호출
  - 실행 결과

```
이 사람의 이름은 홍길동 이고 나이는 20 입니다.
홍길동 은 주민등록번호가 890608-1234567 이고
학번이 0820543 이며 이번 성적은 A 입니다.
```

super 키워드와 상위 클래스의 생성자 호출

- super 키워드를 이용하여 상위 클래스의 생성자 호출 가능
- super() 문은 생성자 첫문장에 자동 정의
  - 상위클래스의 기본 생성자 자동 호출

```
class Employee {
    String   name;
    float    salary;
    Employee() {
        name="unknown";
        salary = 0.0;
    }
}
```

```
class Manager extends Employee {
    Manager(String   n, float   f) {
        name = n;
        salary = f;
    }
    // .....
```

Manager 클래스의 생성자 첫 문장은 자동으로 super();를 포함하여 Employee() 생성자 자동 호출

super 키워드와 상위 클래스의 생성자 호출

- 상위클래스의 매개변수를 정의한 생성자 호출은 명시적으로 super()를 호출하되 정의된 생성자의 매개변수를 전달

```
class Employee {
    String    name;
    float     salary;
    Employee(String    name, float    f) {
        this.name = name;
        this.salary = f;
    }
}
class Manager extends Employee {
    String    department[];
    Manager(String    name, float    f) {
        super(namef);
        department = new String[4];
    }
}
```

사용자가 정의한 생성자.
기본 생성자는 없으므로 하위 클래스에서 super(); 문장 사용 시 오류 발생.

Employee(String) 형태의 생성자 호출.
생성자의 첫 문장으로만 사용 가능

super 키워드와 상위 클래스의 생성자 호출

- 프로그램 6-9 실습
  - SuperConstructorTest.java 작성
  - 상위클래스 생성자 명시적 호출
  - 실행 결과

```
C:₩JAVA>java SuperConstructorTest
클래스 Super의 생성자 수행
클래스 Sub의 생성자 수행
64.0
1000.0
```

객체의 형변환과 다형성

- 객체의 형변환이란
- 객체의 자동 형변환
- 객체의 명시적 형변환
- 객체의 형변환과 사용 멤버들

객체의 형변환이란

- 객체간의 타입 변환은 대입(=) 문장에서 필요
- 객체간의 형변환 필요한 경우

```
객체참조변수1 = 객체참조변수2

클래스타입1 객체참조변수1 = new 클래스타입2()
```

  - 객체참조변수1과 객체참조변수2의 타입 동일하지 않은 경우
  - 클래스타입2의 객체와 객체참조변수1의 타입 동일하지 않은 경우

객체의 자동 형변환

* 객체간의 형변환 규칙

```
객체참조변수1 = 객체참조변수2
```

  * 객체참조변수1이 객체참조변수2의 상위클래스인 경우 자동 형변환

```
class Parent {}
class Child1 extends Parent{}
class Child2 extends Parent{}
```

```
Parent  p;
Child1 c1 = new Child1();
p = c1;                          // 라인 1.
```

```
Parent  p = new Child1();
```

객체의 명시적 형변환

* 하위클래스 타입 변수가 상위클래스 타입 객체 참조하면 컴파일 오류

```
class Parent {}
class Child1 extends Parent{}
class Child2 extends Parent{}
```

```
Parent  p = new Parent();
Child1 c1 ;
c1 =  p;                    // 라인 1.
```

* 하위클래스 타입 변수가 상위클래스 타입 객체 참조시 형변환하면 실행 오류

```
Parent  p = new Parent();
Child1 c1 ;
c1 = (Child1) p;            // 라인 1.
```

객체의 명시적 형변환

- 상속 관계가 없는 클래스 타입의 객체 참조하면 컴파일 오류

```
class Parent {}
class Child1 extends Parent{}
class Child2 extends Parent{}
```

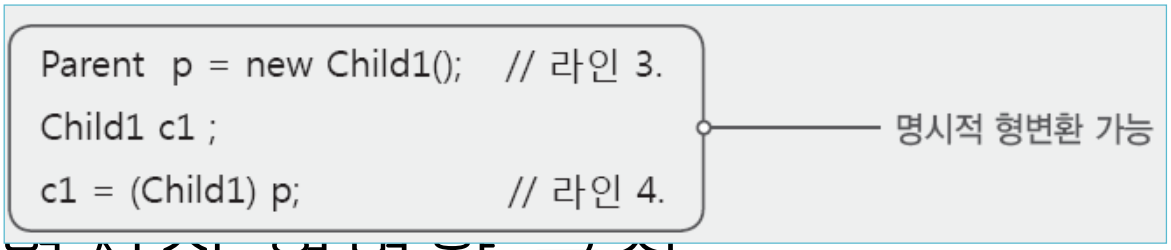
```
Child1 c1 = new Child1();
Child2 c2 ;
c2 = (Child2) c1;            // 라인 1.
```

- 하위클래스 타입 변수가 상위클래스 타입 객체 참조시 형변환하면 실행 오류

```
Parent  p = new Parent();
Child1 c1 ;
c1 = (Child1) p;             // 라인 1.
```

객체의 명시적 형변환

* 명시적 형변환 가능한 경우



* 명시적 형변환 규칙
  * 하위 클래스 타입 변수가 상위 클래스 타입 객체를 참조하는 문장의 경우, 상위 클래스 타입의 변수(라인 3에서의 p)가 변환될 타입의 클래스(라인 3에서는 Child1)나 또는 변환될 타입의 클래스의 하위 클래스타입의 객체를 정확히 갖고 있어야 한다.

객체의 형변환과 사용 멤버들

```
class Employee {
    ......
    int  empBonus;
    void calcSalary() {
        salary = base + empBonus;
    }
}
class Manager extends Employee {
    int  manBonus;
    void setDepartment() {
        department = "Manager";
    }
}
```

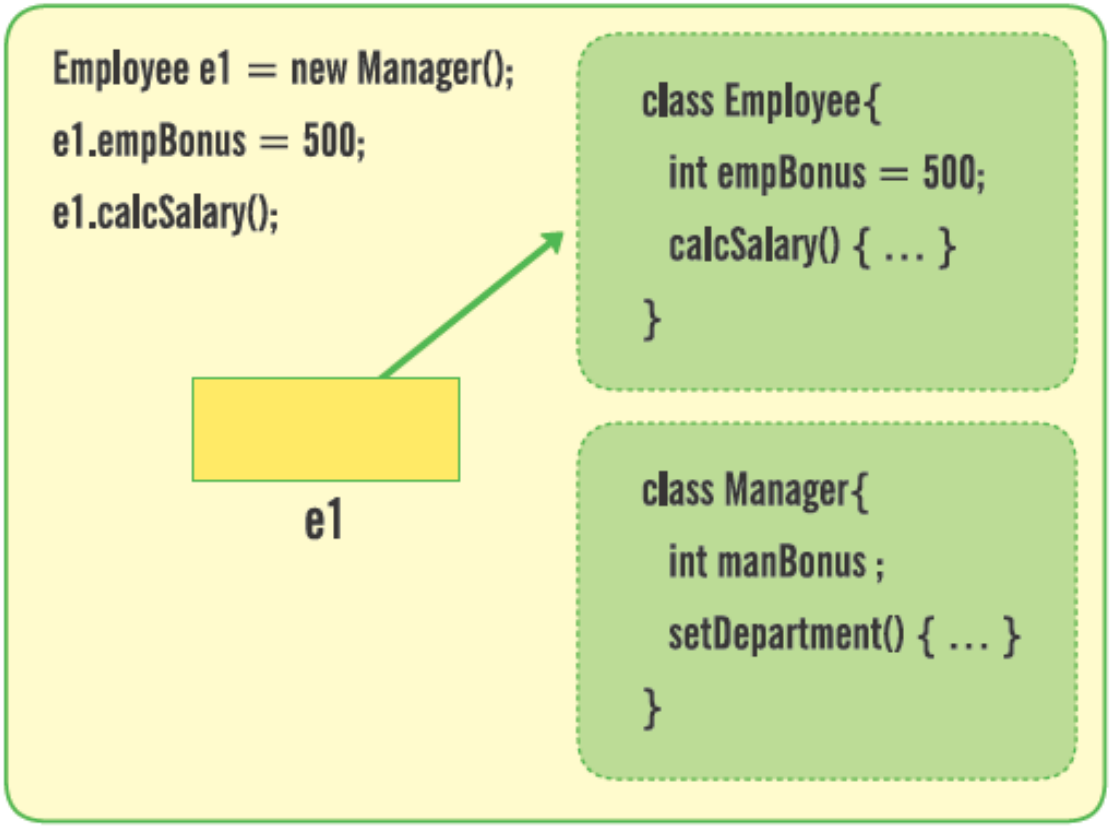
```
Employee e1 = new Manager();  //자동 형변환
```

Employee 클래스의 멤버 변수와 non-overriding 메소드만 사용 가능

```
e1.empBonus = 500;   // 사용 가능
e1.calcSalary();     // 호출 가능. salary = base + empBonus 계산
e1.manBonus = 700;   // 사용 불가능. 오류 발생
e1.setDepartment();  // 호출 불가능. 오류 발생
```

객체의 형변환과 사용 멤버들

* 자동 형변환 시의 메모리 구조



객체의 형변환과 사용 멤버들

```
class Employee {
    ......
    void calcSalary() {
        salary = base + empBonus;
    }
}
class Manager extends Employee {
    ......
    void calcSalary() {
        salary = salary + empBouns + manBonus;
    }
}
```

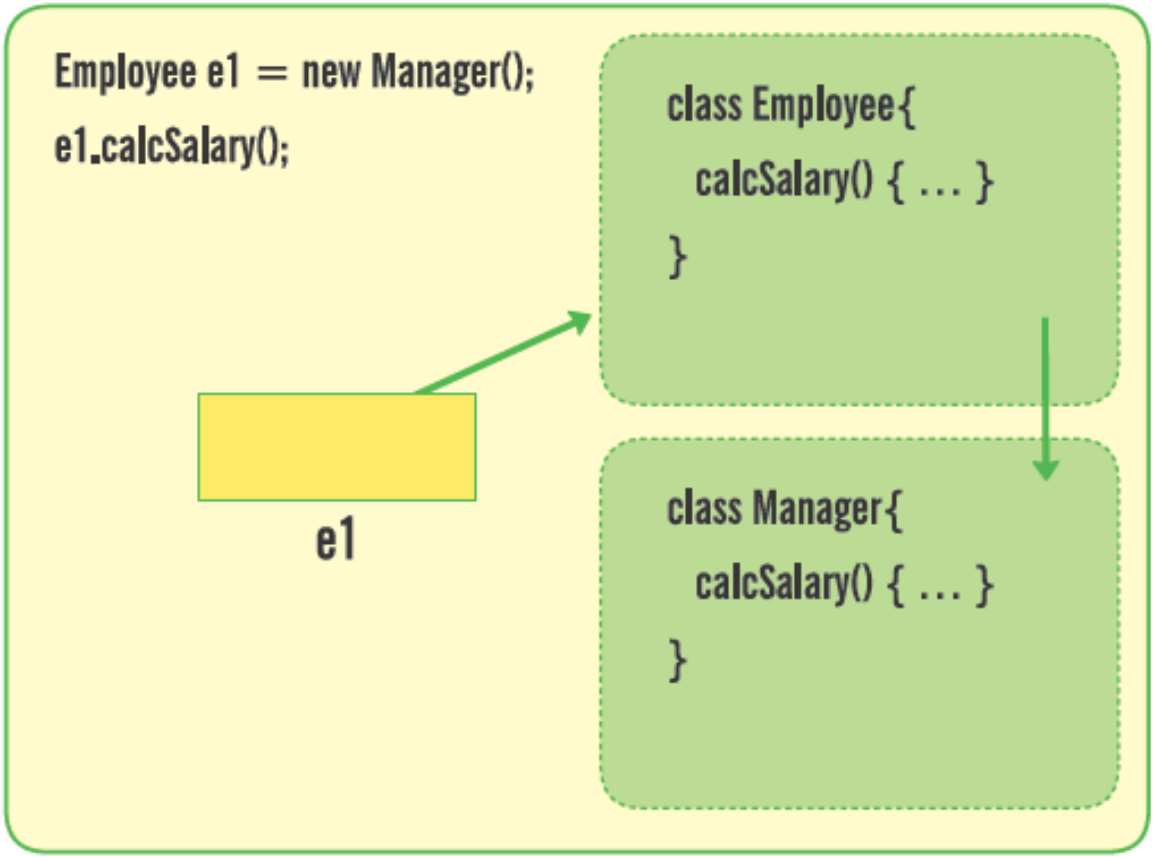
```
Employee e1 = new Manager();  //자동 형변환
```

Manager 클래스의 overriding 메소드 사용 가능

```
e1.calcSalary();      // Manager 클래스의 overriding 된 메소드 실행
```

객체의 형변환과 사용 멤버들

* calcSalary() 호출 시의 메모리 구조



객체의 형변환과 사용 멤버들

* 명시적 형변환하면 Manager 클래스의 멤버들 사용 가능

```
Manager man = (Manager)e1; //e1 변수를 명시적 형변환하여 Manager 타입으로 변경
man.manBonus = 700;        //Manager 클래스에 정의된 변수이므로 사용 가능.
man.setDepartment();       //Manager 클래스에 정의된 메소드이므로 호출 가능.
```

* instanceof 연산자 이용하여 객체 타입 알 수 있음

```
객체참조변수이름 instanceof 클래스이름
```

```
if (emp instanceof Manager) { //emp 변수의 Manager 타입의 객체 참조 여부 조사
    Manager man = (Manager)emp; //true인 경우 Manager 타입으로 명시적 형변환
```

객체의 형변환과 사용 멤버들

- 프로그램 6-10 실습
  - CastingTest.java 작성
  - 객체 형변환후 메소드 호출
  - 실행 결과

```
C:₩JAVA>java CastingTest
salary = 1500.0
Manager 의 salary = 2400.0
```

인터페이스란

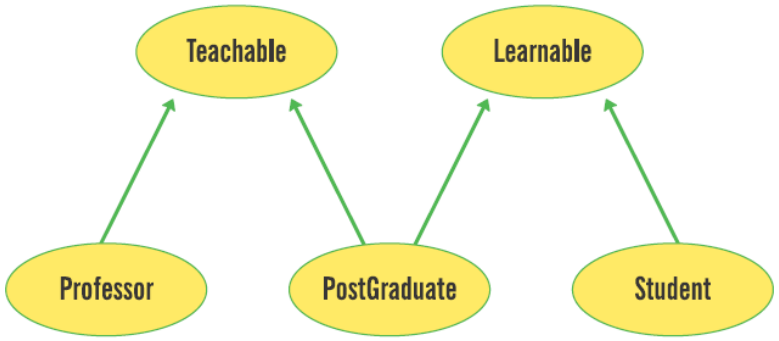
- 인터페이스의 개념과 기능의 통일
- 인터페이스의 특징

인터페이스의 개념과 기능의 통일

- 인터페이스의 멤버 변수는 모두 상수
- 인터페이스의 메소드는 추상 메서드
- 메소드 정의 표준화하는 일종의 틀 제공
- 표준화된 메소드는 각각의 다른 클래스들에서 다른 내용으로 구현

인터페이스의 개념과 기능의 통일

* 다중 상속하는 클래스 예



* 인터페이스의 특징
  * public static final 변수만 포함
  * public abstract 메소드만 포함
  * 생성자 비포함

인터페이스의 개념과 기능의 통일

- 객체 생성 불가능
- super 키워드 사용 불가능
- 구현된 것은 아무 것도 없고 밑그림만 그려진 기본 설계도
- 다중 상속 지원
- 하위 클래스는 메소드 overriding하여 정의

인터페이스의 특징

인터페이스 정의 방법

```
public interface 인터페이스이름 [extends 상위인터페이스1, ...] {
    // 인터페이스 멤버 부분
    public final static type 변수이름;
    ......
    public abstract returnType 메소드이름([매개변수]);
    ......
}
```

인터페이스 선언 예(modifier 생략 가능)

```
interface Teachable {
    public final static String JOB = "가르치는 사람";
    public abstract void teach();
}
```

```
interface Teachable {
    String JOB = "가르치는 사람";
    void teach();
}
```

인터페이스의 활용

- 인터페이스 구현과 implements 키워드
- 인터페이스와 다형성

인터페이스 구현과 implements 키워드

- implements 키워드 사용
- 여러 개의 인터페이스 지정 가능

```
[modifiers] class 클래스이름 [extends 상위클래스이름]
            [implements 인터페이스이름1, 인터페이스이름2, ...] {
    // 클래스 멤버 부분
    // 구현하고자 하는 인터페이스에서 요구하는 메소드 overriding 필요.
        :
}
```

인터페이스 구현과 implements 키워드

* 인터페이스와 클래스의 예

```
interface Teachable {
    String JOB = "가르치는 사람";
    void teach();
}
```

```
interface Learnable {
    String TASK = "배우는 사람";
    void learn();
}
```

```
class PostGraduate implements Teachable, Learnable{    //라인 1
    public void teach(){                               //라인 2
        System.out.println("조교가 가르치다");
    }

    public void learn(){                               //라인 3
        System.out.println("대학원생이 배우다");
    }

}
```

인터페이스와 다형성

* 인터페이스를 구현한 클래스 타입의 객체 생성하여 인터페이스 타입으로 형변환 가능

```
Teachable t = new PostGraduate();   //라인 1
Learnable l = new PostGraduate();   //라인 2
```

* overriding 메소드 호출 가능

```
Teachable t = new PostGraduate();   //라인 1
t.teach();                          //라인 2
t.learn();                          //라인 3
```

인터페이스와 다형성

- instanceof 연산자 이용하여 객체의 형변환 가능 여부 판단

```
t  intanceof  Teachable
```

- 인터페이스를 구현한 클래스 타입으로 명시적 형변환하면 원래의 클래스에 포함된 메소드 사용 가능

```
Teachable t = new PostGraduate();   //라인 1
PostGraduate p = (PostGraduate)t;   //라인 2
```

인터페이스와 다형성

- 프로그램 6-11 실습
  - InterfaceTest.java 작성
  - 두개의 인터페이스와 구현 클래스
  - 실행 결과

```
C:₩JAVA>java InterfaceTest
교수가 가르치다
대학원생이 가르치다
학부생이 배우다
대학원생이 배우다
```

정리

- 상속은 클래스의 코드를 재사용하는 방법으로, 하위 클래스는 상위 클래스의 모든 요소를 자동으로 상속받아 포함하므로, 새로운 요소만 추가하여 정의합니다.
- 상위 클래스에 정의된 메소드와 동일한 이름의 메소드를 하위 클래스에서 재정의할 수 있게 하는 것이 바로 메소드 overriding 입니다.
- 상위 클래스에 정의된 메소드와 동일한 이름의 메소드를 하위 클래스에서 재정의할 수 있게 하는 것이 바로 메소드 overriding 입니다.

정리

- 메소드 overriding의 규칙
  - 메소드 이름, 리턴 타입, 매개변수의 선언은 반드시 같아야 합니다.
  - 접근 modifier는 하위 클래스의 접근 범위가 넓어지는 쪽으로는 변경이 가능합니다.
  - 더 많은 종류의 예외(exception)를 throws로 선언하여 던질 수는 없습니다.
- final modifier는 상수, 메소드 overriding 불가능, 상속 불가능을 표현할 수 있는 modifier입니다.

정리

- abstract modifier는 메소드 overriding과 클래스 상속으로만 사용할 수 있는 modifier입니다.
- super 키워드는 상위 클래스 타입의 객체를 참조하는 키워드입니다.
- 생성자 첫 문장은 자동으로 super(); 문장이 포함되어 상위 클래스의 기본 생성자를 호출하며, 명시적으로 다른 생성자를 호출하기 위해 super()를 이용하는 것도 가능합니다.

정리

- 객체를 참조하는 참조형 변수 간에는 형변환이 가능하며 형변환 규칙은 다음과 같습니다.
  - 상속 관계의 클래스들만 형변환 가능.
  - 하위 클래스 타입의 객체를 상위 클래스 타입으로 자동 형변환 가능.
  - 자동 형변환한 후에 다시 원래의 하위 클래스 타입으로 명시적 형변환 가능.
- 서로 다른 클래스 간에 공통적인 기능을 정의할 때 인터페이스 내부에 공통 기능을 선언하고 인터페이스를 구현하는 하위 클래스들에서 해당 메소드의 내용을 구현하도록 합니다.

정리

- 인터페이스의 특징은 다음과 같이 정리할 수 있습니다.
  - 인터페이스의 멤버 변수는 public static final 변수만 포함합니다.
  - 인터페이스의 모든 메소드는 public abstract 메소드만 포함합니다.
  - 생성자는 포함하지 않습니다.
- 인터페이스를 구현하는 클래스는 implements 키워드를 사용하고 인터페이스 내의 메소드를 overriding하여 활용할 수 있습니다.
- 인터페이스를 구현한 하위 클래스에서는 super 키워드로 인터페이스 객체를 참조하는 것이 불가능합니다.