|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 실습9 | 학번 | 20163109 | 이름 | 박재효 |
| 결과보고서 | 학년 / 분반 | 1 / 02 | 제출일 | 2016. 11. 16 |

▶ 상속을 통한 새로운 타입 정의

1.

**public** **class** Commission **extends** Hourly{

**private** **double** totalSales;

**private** **double** rateOfCm;

**public** Commission (String eName, String eAddress, String ePhone,

String socSecNumber, **double** hoursWorked, **double** rateOfCm){

**super**(eName, eAddress, ePhone, socSecNumber, hoursWorked);

**this**.rateOfCm = rateOfCm ;

}

**public** **void** addSales (**double** totalSaless){

**this**.totalSales += totalSaless;

}

**public** **double** pay()

{

**double** payment = **super**.pay() + rateOfCm\*totalSales;

totalSales = 0;

**return** payment;

}

**public** String toString(){

String result = **super**.toString();

result += "\nTotal Sales: " + totalSales;

**return** result;

}

}

2. 정상실행

3.

**public** **void** payday ()

{

**double** amount;

**for** (StaffMember staffList : staffList)

{

System.***out***.println (staffList);

amount = staffList.pay(); // polymorphic

**if** (amount == 0.0)

System.***out***.println ("Thanks!");

**else**

System.***out***.println ("Paid: " + amount);

System.***out***.println ("-----------------------------------");

}

}

4. 네

▶ Overriding과 Overshadowing

1.

|  |
| --- |
| Parent |
| printID()  showMessage() |

↑

|  |
| --- |
| Son |
| printID() |

↑

|  |
| --- |
| GrandSon |
| printID()  showMessage()  doGrandSon()  examineObjects() |

2.

grandSon.printID(); : I am a GrandSon.

>>> grandSon 클래스에서 printID() 상속 받는다.

son.printID(); : I am a GrandSon.

>>> grandSon을 son에 오버라이딩 하여서 son.printID()를 하면 위의 출력력값이 나온다.

super.printID(); : I am a Son.

>>> 현재 클래스인 GrandSon 클래스가 상속받은 상위 클래스인 son에서 printID()를 상속받는다.

showMessage(); : Grand son message: I am 5years old.

>>> 현재 클래스인 GrandSon 클래스에서 showMessage()를 사용한다.

super.showMessage(); : Parent message: I am 60years old.

>>> 현재 클래스인 GrandSon 클래스가 상속받은 상위 클래스인 son에 showMessage()가 없어 son의 상위 클래스인 parent에서 showMessage()를 사용한다.

3.

에러가 발생한다.

grandSon에 doGrandSon()이 없다.

4.

에러가 발생한다.

Super를 중첩해서 그렇다.

5.

age = 15

>>> grandson.examineObject(15); 에서 15로 인자를 넣어서.

this.age = 5

>>> this가 강제로 현재 클레스에서의 age를 받는다.

Super.age = 35

>>> 현재 클래스가 상속 받은 상위 클래스인 son에서의 age값을 가져온다.

son.age = 35

>>> 변수는 static으로 선언을 하지 않으면 오버라이딩이 되지 않는다.

parent.age = 60

>>> 변수는 static으로 선언을 하지 않으면 오버라이딩이 되지 않는다.

((Son) this).age = 35

>>> son 클래스의 age값을 가져온다.

((Parent) this).age = 60

>>> parent 클래스의 age 값을 가져온다.

6.

에러가 난다.

Super를 중첩해서 그렇다.

7. 네

▶ Java 객체의 비교

1.

alice1 == alice2: true

>>> alice1과 alice2의 주소 값이 같음으로 true가 출력이 된다.

alice1 == alice3: false

>>> alice1과 alice3의 주소 값이 다름으로 false가 출력이 된다.

alice1.equals(alice3): true

>>> alice1과 alice3의 내용이 같아서 true 가 출력이 된다.

alice1.equals(bob): false

>>> 값이 달라서 false이다.

2.

bob.toString() :Employee[name=Bob Brandson,salary=50000.0,hireDay=Sun Oct 01 00:00:00 KST 1989]

>>> bob은 Employee에 속해 있어 employee의 toString에 값을 대입하여 출력을 한다.

boss.toString() :Manager[name=Carl Cracker,salary=80000.0,hireDay=Tue Dec 15 00:00:00 KST 1987][bonus=5000.0]

>>> boss는 Manager에 속해져 있어 employee의 toString에 <“bonus=” +bonus>를 추가한 값이 출력이 된다.

3.

carl.equals(boss) :false

>>> boss.setBonus(5000)으로 인해 둘의 값이 달라서 false가 나온 것이다.

4.

alice1.hashCode() :330976739

alice3.hashCode() :330976739

bob.hashCode() :908481687

carl.hashCode() :339710272

해시코드를 출력한다.

Alice1과 alice3는 객체의 내용이 같아서 해시코드가 동일하다.

5. 네

▶ Inheritance

1.

|  |
| --- |
| Card |
| card()  getname()  isExpired()  fomat() |

↗ ↑ ↖

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| IDCard |  | CallingCard |  | DriverLicense |
| IDCard() |  | CallingCard() |  | DriverLicense() |

▶ Calling the Superclass Constructor

2.

**class** IDCard **extends** Card{

**private** String idNumber;

**public** IDCard(String n, String id){

**super**(n);

idNumber = id;

}

}

**class** CallingCard **extends** Card{

**private** **int** pin;

**private** **int** cardNumber;

**public** CallingCard(String n, **int** cardNum, **int** pinNum){

**super**(n);

cardNumber = cardNum;

pin = pinNum;

}

}

**class** DriverLicense **extends** Card{

**private** **int** expirationYear;

**public** DriverLicense(String n, **int** expYear){

**super**(n);

expirationYear = expYear ;

}

}

▶ Overriding Methods

3.

**class** IDCard **extends** Card{

**private** String idNumber;

**public** IDCard(String n, String id){

**super**(n);

idNumber = id;

}

}

**class** CallingCard **extends** Card{

**private** **int** pin;

**private** **int** cardNumber;

**public** CallingCard(String n, **int** cardNum, **int** pinNum){

**super**(n);

cardNumber = cardNum;

pin = pinNum;

}

**public** String format(){

String result = **super**.format() + ", Card Number: " + cardNumber + ", PIN: " + pin;

**return** result;

}

}

**class** DriverLicense **extends** Card{

**private** **int** expirationYear;

**public** DriverLicense(String n, **int** expYear){

**super**(n);

expirationYear = expYear ;

}

**public** String format(){

String result = **super**.format() + ", Expration Year of Driver License:" + expirationYear;

**return** **super**.format();

}

}

4.

**class** Billfod{

**private** Card card1;

**private** Card card2;

**public** **void** addCard(Card c){

**if** (card1 == **null**) {

card1 = c;

}

**else** **if** (card2 == **null**){

card2 = c;

}

}

**public** String formatCards(){

String result = card1.format() + "\n" +card2.format();

**return** result;

}

}

5.

**class** BillfodTest {

**public** **static** **void** main (String args[]){

Billfod bf = **new** Billfod();

Card card1 = (Card) **new** CallingCard("a", 123, 456);

Card card2 = (Card) **new** CallingCard("b", 789, 999);

bf.addCard(card1);

bf.addCard(card2);

System.***out***.println(bf.formatCards());

}

}

6.

Card holder: a, Card Number: 123, PIN: 456

Card holder: b, Card Number: 789, PIN: 999

7.

객체를 하나로 통일시키기 위해서

8.

**public** **boolean** isExpired(){

GregorianCalendar calendar = **new** GregorianCalendar();

**int** currentYear = calendar.get(Calendar.***YEAR***);

**if** (currentYear > expirationYear) {

**return** **false**;

}

**else** {

**return** **true**;

}

}

9.

메쏘드를 구현하지 않고 그대로 둔다.

10.

**public** **int** getExpiredCardCount(){

**int** count = 0;

**if**( card1 **instanceof** DriverLicense){

**if**(!(card1.isExpired()))

{

count++;

}

}

**if**( card2 **instanceof** DriverLicense){

**if**(!(card2.isExpired())){

count++;

}

}

**return** count;

}

11.

Billfod bf2 = **new** Billfod();

Card card3 = **new** CallingCard("Alice", 4457, 9964);

Card card4 = **new** DriverLicense("Tom", 2014);

bf2.addCard(card3);

bf2.addCard(card4);

System.***out***.println(bf2.getExpiredCardCount());

▶ The toString method 12.

Card ::

**public** String toString(){

String result = "name=" + name;

**return** "[" + result + "]";

}

IDCard ::

**public** String toString(){

String result = **super**.toString() + ", ID Number=" + idNumber;

**return** "[" + result + "]";

}

CallingCard ::

**public** String toString(){

String result = **super**.toString() + ", Card Number =" + cardNumber +

", PIN="+pin;

**return** "[" + result + "]";

}

DriverLicense ::

**public** String toString(){

String result = **super**.toString() + ", Expiration Year=" + expirationYear;

**return** "[" + result + "]";

}

Billfod ::

**public** String toString(){

**if**( card1 == **null**){

**return** "Billfold";

}

**else** **if**( card2 == **null**){

**return** "Billfold" + **this**.card1.getName();

}

**return** "Billfold=" + **this**.card1.getName() + ", "+ **this**.card2.getName();

}

▶ The equals method 13.

**public** **boolean** equals(Card c1, Card c2){

**if** (c1.getClass().getName().equals(c2.getClass().getName())

&& c1.toString().equals(c2.toString())) {

**return** **true**;

}

**else** {

**return** **false**;

}

}

▶ Protected access

14.

1. 쉽다. Private와 달리 protected는 동일한 패키지 속에 속하는 클래스에서 접근이 가능하기 때문이다. 또, 자신을 상속받는 클래스에서 접근이 가능하다.

2. protected로 바꿔주면 앞으로 자식 클래스를 오버라이딩해서 바꾸어야 할 경우에 고치기가 더 쉬워진다.