10장. 상속

Section 1 생성자
Section 2 생성자 오버로딩
Section 3 예약어 this
Section 4 메소드
Section 5 메소드 오버로딩
Section 6 메소드에 값 전달 기법

처음시작하는 JAVA프로그래밍 Essential Course

▪ 학습 목표

- 절차 지향에서의 재사용 방법과 상속의 개요를 학습합니다.
- 상속의 효과와 특징에 대해 학습하고, 자바의 상속에 관해 학습합니다.
- 클래스 상속 시 한정자 지정에 따른 접근 여부에 관해 학습합니다.
- 생성자와 상속에 관해 학습합니다.
- 상속 관계에서 묵시적 생성자와 명시적 생성자의 차이를 학습합니다.
- 예약어 super에 관해 학습합니다.
- Object 클래스의 속성과 기능에 관해 학습합니다.
- 예약어 final에 관해 학습합니다.

1-1 모듈의 재사용과 객체지향

● 객체지향 이전에 모듈의 재사용

- 모듈을 복사(copy & paste)하여 일부 내용을 변경하고, 추가하여 사용
- 비슷한 모듈이 양산되어, 나중에는 원하는 모듈을 발견하기 어려운 문제를 가지고 있다

그림 10-1 절차 지향에서의 모듈 재사용

1-1 모듈의 재사용과 객체지향

● 객체지향에서는 상속으로 문제를 해결

- 상속은 상위 클래스의 속성과 기능을 하위 클래스 상속시키고, 하위 클래스에는 새로운 기능과 속성을 추가하면 된다
- 자바에서의 상속은 확장(extends)의 개념
  - 하위 클래스 = 상위 클래스 + 하위 클래스에 추가된 기능과 속성

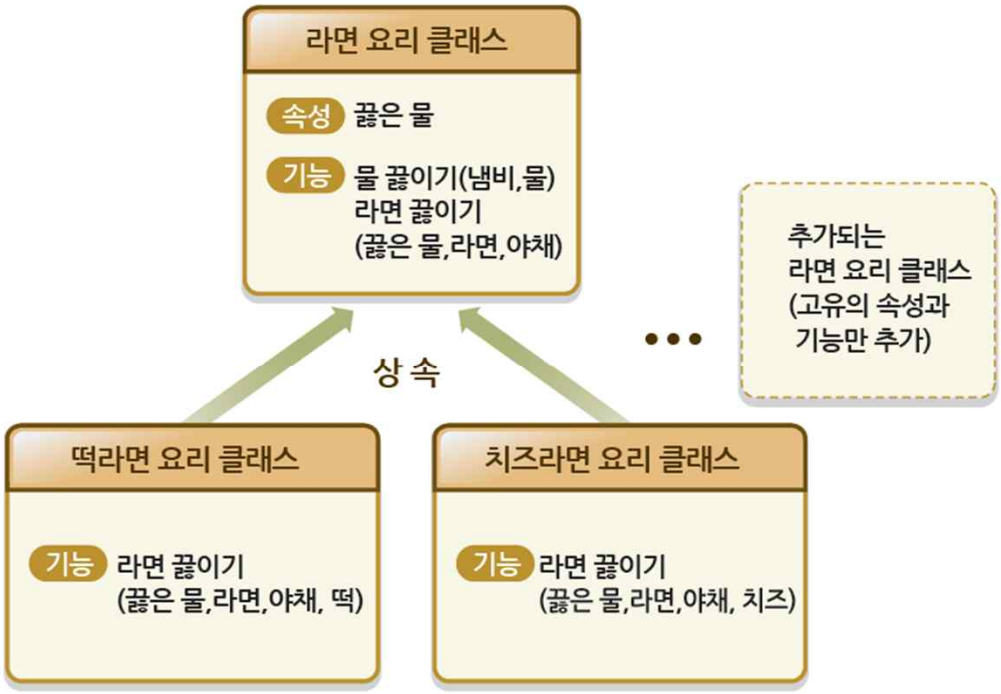


그림 10-2 객체 지향에서의 상속

상속의 개념을 이용하면, 새로운 라면 요리가 개발되더라도, 코드의 중복 없이 추가적인 속성과 기능만을 추가하여 쉽게 만들 수 있습니다.

1-2 상속의 효과와 자바 상속의 특징

● 상속의 효과

- 클래스 검색이 쉬움
  - 상속을 통하여 클래스를 체계화할 수 있습니다.
- 클래스 확장이 쉬움
  - 상속을 통하여 새로운 클래스를 생성하기가 용이합니다
- 클래스 변경이 쉬움
  - 상속을 통하여 기존 클래스의 기능을 쉽게 변경할 수 있습니다.

1-2 상속의 효과와 자바 상속의 특징

● 자바 상속의 특징

- 다중 상속을 허용하지 않습니다.
- 자바 모든 클래스는 Object 클래스로부터 상속됩니다.
  - 자바의 최상위 클래스는 Object 클래스입니다. Object 클래스는 명시적으로 상속하지 않아도 묵시적 상속이 이루어집니다.

1-3 자바의 상속

- **자바 언어에서 상속은 객체 지향 언어의 장점인 모듈의 재사용과 코드의 간결성을 제공하는 중요한 특성**
  - 클래스 선언 시 상속을 지정하기 위해 extends라는 예약어를 사용

[형식] 상속

```
class 클래스 이름 extends 상위클래스 이름 {
    .....추가되는 멤버 변수
    .....생성자(생성자는 상속되지 않는다)
    .....추가되는 메소드
}
```

1-3 자바의 상속

- 예제 10.1

예제 10.1 CookingRamyon1.java

```
01: import java.util.Scanner;
02: class Ramyon {
03:     protected String boiled_pot;
04:    protected String boilwater(String elecpot, String water) {
05:         System.out.println(elecpot +"에 " + water +"을 넣어
    끓인다(Ramyon클래스)");
06:         return "끓은 물";
07:     }
08:     public void cookRamyon(String ramyon, String vegetable, int time) {
09:         boiled_pot = boilwater("전기 냄비","물");
10:         System.out.println(boiled_pot + "에 " + ramyon + "과 " + vegetable +
    "를 넣고 " + time + "분간 끓인다(Ramyon클래스)");
11:         System.out.println("일반 라면 요리 완료(Ramyon클래스)");
12:     }
13: }
14:
15: class RiceRamyon extends Ramyon {
16:     public void cookRamyon(String ramyon, String vegetable, String rice, int
    time) {
17:         boiled_pot = boilwater("전기 냄비","물");
18:         System.out.println(boiled_pot + "에 " + ramyon + "과 " + vegetable +
    "와 " + rice + "을 넣고 " + time + "분간 끓인다(RiceRamyon클래스)");
19:         System.out.println("떡라면 요리 완료(RiceRamyon클래스)");
20:     }
21: }
22:
23: class CheeseRamyon extends Ramyon {
24:     public void cookRamyon(String ramyon, String vegetable, String cheese, int
    time) {
25:         boiled_pot = boilwater("전기 냄비","물");
```

상속되어 사용될 속성이므로 protected로 선언

상속되어 사용될 메소드이므로 protected로 선언

라면을 끓이는 메소드 선언

물을 끓이는 메소드 호출

Ramyon 클래스의 하위 클래스로 RiceRamyon 클래스 생성

상위 클래스와 같은 이름의 메소드 선언(매개 변수의 개수와 형이 다르다)

상속받은 boilwater() 메소드 호출

Ramyon 클래스의 하위 클래스로 CheeseRamyon 클래스 생성

상위 클래스와 같은 이름의 메소드 선언(매개 변수의 개수와 형이 다르다)

● 예제

```
26:         System.out.println(boiled_pot + "에 " + ramyon + "과 " + vegetable +
    "과 " + cheese + "를 넣고 " + time + "분간 끓인다(CheeseRamyon클래스)");
27:         System.out.println("치즈라면 요리 완료(CheeseRamyon클래스)");
28:     }
29: }
30:
31: public class CookingRamyon1 {
32:     public static void main(String args[]) {
33:         int s;
34:         do {
35:             System.out.print("만들고 싶은 요리를 입력하세요(1:일반 라면,
    2:떡라면,3:치즈라면,99:종료) : ");
36:             Scanner stdin = new Scanner(System.in);
37:             s = stdin.nextInt();
38:             if (s == 1) {
39:                 Ramyon r = new Ramyon();
40:                 r.cookRamyon("신라면", "파", 5);
41:             } else if (s == 2) {
42:                 RiceRamyon rr = new RiceRamyon();
43:                 rr.cookRamyon("안성탕면", "양파", "떡국떡", 7);
44:                 rr.cookRamyon("칼국수 라면", "쪽파", 4);
45:             } else if (s == 3) {
46:                 CheeseRamyon cr = new CheeseRamyon();
47:                 cr.cookRamyon("진라면", "버섯", "슬라이스치즈", 6);
48:             }
49:         } while ( s != 99);
50:
51:     }
52: }
```

cookRamyon() 메소드 호출

RiceRamyon 클래스의 cookRamyon() 메소드 호출

떡라면 객체를 이용하여 일반 라면을 끓인다.

CheeseRamyon 클래스의 cookRamyon() 메소드 호출

실행 결과

```
만들고 싶은 요리를 입력하세요(1:일반라면,2:떡라면,3:치즈라면,99:종료) : 1
전기 냄비에 물을 넣어 끓인다(Ramyon클래스)
끓은 물에 신라면과 파를 넣고 5분간 끓인다(Ramyon클래스)
일반 라면 요리 완료(Ramyon클래스)
만들고 싶은 요리를 입력하세요(1:일반라면,2:떡라면,3:치즈라면,99:종료) : 2
전기 냄비에 물을 넣어 끓인다(Ramyon클래스)
끓은 물에 안성탕면과 양파와 떡국떡을 넣고 7분간 끓인다(RiceRamyon클래스)
떡라면 요리 완료(RiceRamyon클래스)
전기 냄비에 물을 넣어 끓인다(Ramyon클래스)
끓은 물에 칼국수라면과 쪽파를 넣고 4분간 끓인다(Ramyon클래스)
일반 라면 요리 완료(Ramyon클래스)
만들고 싶은 요리를 입력하세요(1:일반라면,2:떡라면,3:치즈라면,99:종료) : 3
전기 냄비에 물을 넣어 끓인다(Ramyon클래스)
끓은 물에 진라면과 버섯과 슬라이스치즈를 넣고 6분간 끓인다(CheeseRamyon클래스)
치즈라면 요리 완료(CheeseRamyon클래스)
만들고 싶은 요리를 입력하세요(1:일반라면,2:떡라면,3:치즈라면,99:종료) : 99
```

- 자바에서 한정자를 사용할 수 있는 대상
  - 클래스
  - 멤버 변수
  - 메소드

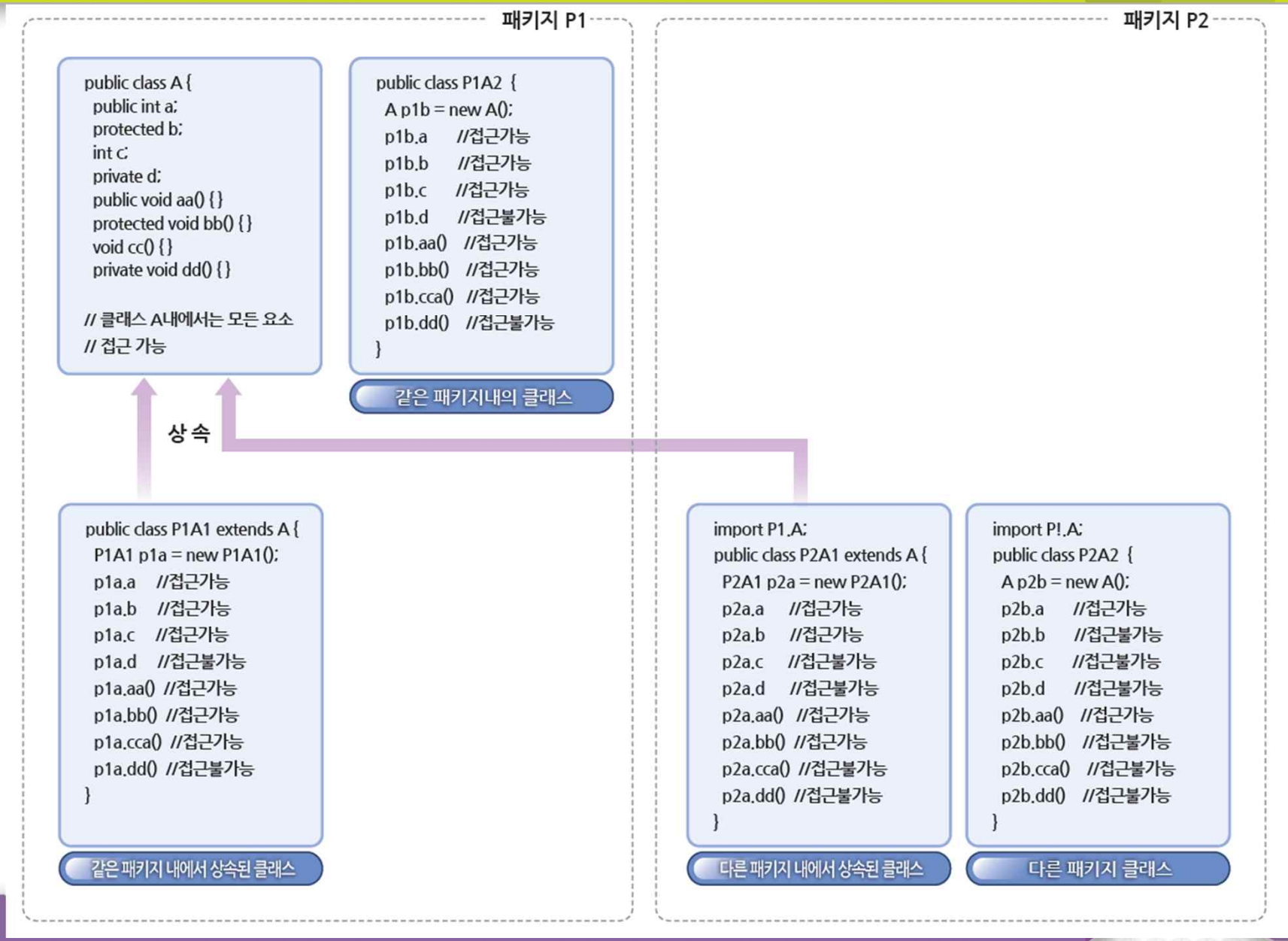
  * 생성자(생성자의 한정자는 별 의미가 없음, public이나 붙이지 않거나, private가 사용될 수 있다

- **public, 한정자 없음, private외에 추가로 protected 한정자가 있다**
  - public : 동일 패키지나 상속 관계 등에 상관없이 모든 클래스에서 사용 가능합니다.
  - protected : 동일한 패키지이면 상속 여부에 상관없이 사용 가능하고, 다른 패키지의 클래스에서는 상속된 경우에만 사용 가능합니다.
  - default(한정자 없음) : 동일한 패키지이면 상속 여부에 상관없이 사용 가능합니다.
  - private : 어떠한 경우에도 사용이 불가능합니다. 해당 클래스 내부에서만 사용 가능합니다.

표 10-1 한정자에 따른 접근 가능 여부

접근 한정자	동일한 클래스	같은 패키지 내의 모든 클래스 (상속 여부에 상관없음)	다른 패키지	
			상속된 클래스	상속되지 않은 클래스
public	O	O	O	O
protected	O	O	O	X
default(한정자 없음)	O	O	X	X
private	O	X	X	X

2 상속과 한정자


패키지 P1
패키지 P2
public class A {
public int a;
protected b;
int c;
private d;
public void aa() {}
protected void bb() {}
void cc() {}
private void dd() {}
// 클래스 A내에서는 모든 요소
// 접근 가능
public class P1A2 {
A p1b = new A();
p1b.a //접근가능
p1b.b //접근가능
p1b.c //접근가능
p1b.d //접근불가능
p1b.aa() //접근가능
p1b.bb() //접근가능
p1b.cca() //접근가능
p1b.dd() //접근불가능
}
같은 패키지내의 클래스
상 속
public class P1A1 extends A {
P1A1 p1a = new P1A1();
p1a.a //접근가능
p1a.b //접근가능
p1a.c //접근가능
p1a.d //접근불가능
p1a.aa() //접근가능
p1a.bb() //접근가능
p1a.cca() //접근가능
p1a.dd() //접근불가능
}
같은 패키지 내에서 상속된 클래스
import P1.A;
public class P2A1 extends A {
P2A1 p2a = new P2A1();
p2a.a //접근가능
p2a.b //접근가능
p2a.c //접근불가능
p2a.d //접근불가능
p2a.aa() //접근가능
p2a.bb() //접근가능
p2a.cca() //접근불가능
p2a.dd() //접근불가능
}
다른 패키지 내에서 상속된 클래스
import P1.A;
public class P2A2 {
A p2b = new A();
p2b.a //접근가능
p2b.b //접근불가능
p2b.c //접근불가능
p2b.d //접근불가능
p2b.aa() //접근가능
p2b.bb() //접근불가능
p2b.cca() //접근불가능
p2b.dd() //접근불가능
}
다른 패키지 클래스

2 상속과 한정자

- 예제 10.2

예제 10.2 InheritanceTest1.java

```
01: class A {
02:     public int aa = 1;
03: }
04: class B extends A {
05:     private int bb = 2;
06:     public int bb() {
07:         return bb;
08:     }
09: }
10: class C extends B {
11:     int cc = 3;
12: }
13: public class InheritanceTest1  {
14:     public static void main(String[] args) {
15:         C objc = new C();
16:         System.out.println("objc객체의 객체 속성 변수 aa의 값은 " + objc.aa);
17:         System.out.println("objc객체의 객체 속성 변수 bb의 값은 " + objc.bb());
18:         //objc.bb에 접근하면 오류 발생
19:         System.out.println("objc객체의 객체속성변수 cc의 값은 " + objc.cc);
20:     }
21: }
```

멤버 변수가 public으로 선언

클래스 A로부터 상속

멤버 변수가 private으로 선언

메소드 bb() 선언. private 변수의 값을 반환

클래스 B로부터 상속

한정자 없이 멤버 변수 선언

private 속성에 적접 접근할 수 없으므로 메소드를 통해 값을 출력

객체의 public 속성에 직접 접근 가능

클래스 C로부터 객체 생성

한정자 없이 사용한 속성에 접근

실행 결과

```
objc객체의 객체 속성 변수 aa의 값은 1
objc객체의 객체 속성 변수 bb의 값은 2
objc객체의 객체 속성 변수 cc의 값은 3
```

● 예제 10.3

예제 10.3 InheritanceTest2.java

```
01: class B1 {
02:     public int x = 500;
03:     public int y = 1000;
04: }
05: class B2 extends B1 {
06:     public String x = "처음 시작하는 자바";
07: }
08: public class InheritanceTest2 {
09:     public static void main(String args[]) {
10:         B2 b2 = new B2();
11:         System.out.println("객체 b2에 들어 있는 x,y 값 : " + b2.x + b2.y);
12:         B1 b1 = new B1();
13:         System.out.println("객체 b1에 들어 있는 x,y 값 : " + b1.x + b1.y);
14:     }
15: }
```

public 멤버 변수 선언

상위 클래스와 같은 이름으로 public x 선언.
상위 클래스의 값은 가려짐

하위 클래스의 x 값 출력. y값은 상속된 값 출력

상위 클래스의 속성값 x,y 출력

실행 결과

```
객체 b2에 들어 있는 x,y 값 : 처음 시작하는 자바1000
객체 b1에 들어 있는 x,y 값 : 5001000
```

- 예제 10.4

예제 10.4 InheritanceTest3.java

```
01: class C1 {
02:     private static int x=100; <---------- 클래스 변수를 private로 선언
03:     public static int y = x;
04:     static int z=x;
05:     public static int cc() { return x;} <------ 클래스 메소드 선언
06: }
07: class C2 extends C1 {
08:     public static String x; <---------------- 클래스 변수로 x 선언
09:     static int y; <------------ 상위 클래스와 같은 이름의 클래스 변수 선언
10: }
11: class InheritanceTest3 {
12:     public static void main(String args[]) {
13:         System.out.println("클래스 메소드 C1.cc() 값 : " + C1.cc());
14:         System.out.println("클래스 변수 C1.y 값 : " + C1.y);
15:         System.out.println("클래스 변수 C1.z 값 : " + C1.z);
16:         System.out.println("클래스 변수 C2.x 값 : " + C2.x);
17:         System.out.println("클래1스 변수 C2.y 값 : " + C2.y);
18:         System.out.println("클래스 변수 C2.z 값 : " + C2.z);
19:         System.out.println("클래스 메소드 C2.cc() 값 : " + C2.cc());
20:         C2.x = "처음 시작하는 자바";
21:         C2.y = 200;
22:         C1.z = 300;//C2.z = 300;도 가능
23:         System.out.println("클래스 변수 C2.x 값 : " + C2.x);
24:         System.out.println("클래스 변수 C2.y 값 : " + C2.y);
25:         System.out.println("클래스 변수 C2.x 값 : " + C2.z);
26:   }
27: }
```

실행 결과

```
클래스 메소드 C1.cc() 값 : 100
클래스 변수 C1.y 값 : 100
클래스 변수 C1.z 값 : 100
클래스 변수 C2.x 값 : null
클래1스 변수 C2.y 값 : 0
클래스 변수 C2.z 값 : 100
클래스 메소드 C2.cc() 값 : 100
클래스 변수 C2.x 값 : 처음 시작하는 자바
클래스 변수 C2.y 값 : 200
클래스 변수 C2.x 값 : 300
```

- **클래스의 상속에서 생성자는 상속되지 않지만, 다음과 같은 특성을 가진다**
  - 상속 관계에서 하위 클래스에 묵시적 생성자가 있는 경우, 하위 클래스로부터 객체가 생성될 때 상위 클래스의 묵시적 생성자가 우선 수행.
  - 상속 관계에서 하위 클래스에 명시적(매개 변수가 있는) 생성자만 있는 경우, 상위 클래스의 묵시적 생성자가 우선 수행.
  - 상속 관계에서 하위 클래스에 명시적 생성자와 묵시적 생성자가 모두 있는 경우, 하위 클래스의 어떠한 생성자가 호출되더라도 상위 클래스의 묵시적 생성자가 우선 수행

3 상속과 생성자

- 예제 10.5

```
01: class DA1 {
02:     public double d1;
03:     public DA1() { <------ 묵시적 생성자 선언
04:         System.out.println("클래스 DA1의 묵시적 생성자 수행");
05:         d1 = 10*10;
06:     }
07: }
08: class DA2 extends DA1 {
09:    public double d2;
10:    public DA2() { <------ 묵시적 생성자 선언
11:         System.out.println("클래스 DA2의 묵시적 생성자 수행");
12:         d2 = 10*10*10;
13:     }
14: }
15: class DA3 extends DA2 {
16:     public double d3;
17:     public DA3() { <------ 묵시적 생성자 선언
18:         System.out.println("클래스 DA3의 묵시적 생성자 수행");
19:         d3 = 10*10*10*10;
20:     }
21: }
22: public class DefaultInheritanceTest1 {
23:       public static void main(String args[]) {
24:           DA3 super1 = new DA3(); <--------클래스 A3로부터 객체 생성. 묵시적 생성자 수행
25:           System.out.println("10의 2제곱 : " + super1.d1);
26:           System.out.println("10의 3제곱 : " + super1.d2);
27:           System.out.println("10의 4제곱 : " + super1.d3);
28:           DA2 super2 = new DA2(); <--------클래스 A2로부터 객체 생성. 묵시적 생성자 수행
29:       }
30: }
```

실행 결과

```
클래스 DA1의 묵시적 생성자 수행
클래스 DA2의 묵시적 생성자 수행
클래스 DA3의 묵시적 생성자 수행
10의 2제곱 : 100.0
10의 3제곱 : 1000.0
10의 4제곱 : 10000.0
클래스 DA1의 묵시적 생성자 수행
클래스 DA2의 묵시적 생성자 수행
```

● 예제 10.6

예제 10.6 DefaultInheritanceTest2.java

```
01: class DB1 {
02:     public double d1;
03:     public DB1() {
04:         System.out.println("클래스 DB1의 묵시적 생성자 수행");
05:         d1 = 10*10;
06:     }
07: }
08: class DB2 extends DB1 {
09:     public double d2;
10:     public DB2() {
11:         System.out.println("클래스 DB2의 묵시적 생성자 수행");
12:         d2 = 10*10*10;
13:     }
14:    public DB2(double d ) {  <---------- 매개 변수가 있는 명시적 생성자 선언
```

● 예제 10.6

```
15:         System.out.println("클래스 DB2의 명시적 생성자 수행");
16:         d2 = d*d*d;
17:     }
18: }
19: class DB3 extends DB2 {
20:     public double d3;
21:     public DB3() {
22:         System.out.println("클래스 DB3의 묵시적 생성자 수행");
23:         d3 = 10*10*10*10;
24:     }
25:    public DB3(double d ) {  <-- 매개 변수가 있는 명시적 생성자 선언
26:         System.out.println("클래스 DB3의 명시적(매개 변수1개) 생성자 수행");
27:        d3 = d*d*d*d;
28:     }
29: }
30: public class DefaultInheritanceTest2 {
31:       public static void main(String args[]) {
32:           DB3 super1 = new DB3(11);  <-- 명시적 생성자를 호출하는 객체 생성
33:           System.out.println("10의 2제곱 : " + super1.d1);
34:           System.out.println("10의 3제곱 : " + super1.d2);
35:           System.out.println("10의 4제곱 : " + super1.d3);
36:       }
37: }
```

실행 결과

```
클래스 DB1의 묵시적 생성자 수행
클래스 DB2의 묵시적 생성자 수행
클래스 DB3의 명시적(매개 변수1개) 생성자 수행
10의 2제곱 : 100.0
10의 3제곱 : 1000.0
10의 4제곱 : 14641.0
```

4-1 메소드 오버라이딩

- **오버라이딩(overriding)**
  - 상속 관계에 있는 클래스들 간에 같은 이름의 메소드를 정의하는 경우
  - 상위 클래스의 메소드와 하위 클래스의 메소드가 메소드 이름은 물론 매개변수의 타입과 개수까지도 같아야 함

  - 9장에서 배운 메소드 오버로딩(Overloading)과 혼동하지 말것
    - 오버로딩 : 같은 클래스 내에서 같은 이름의 생성자나 메소드를 사용하는 경우

- **동일한 클래스 내에서 이루어지는 오버로딩을 중첩(같은 메소드 여러 개 존재)으로 표현한다면, 오버라이딩은 치환(상위 클래스의 메소드를 하위 클래스의 메소드로 교체)으로 표현할 수 있다**

4-1 메소드 오버라이딩

● 예제 10.7

예제 10.7 OverridingTest1.java

```
01: class OIa {
02:     public void show(String str) {
03:         System.out.println("상위 클래스의 메소드 show(String str)수행 " + str);
04:     }
05: }
06: class OIb extends OIa {
07:     public void show() {
08:         System.out.println("하위 클래스의 메소드 show() 수행");
09:     }
10: }
11: public class OverridingTest1 {
12:     public static void main(String args[]) {
13:         OIb oib = new OIb();
14:         oib.show("처음 시작하는 자바");
15:         oib.show();
16:     }
17: }
```

하나의 매개 변수를 가진 show() 메소드 선언

상속된 클래스에서 매개 변수 없는 show() 메소드 선언

각각에 적합한 메소드가 호출된다.

실행 결과

```
상위 클래스의 메소드 show(String str)수행 처음 시작하는 자바
하위 클래스의 메소드 show() 수행
```

- 예제 10.8

예제 10.8 OverridingTest2.java

```
01: class Ramyon1 {
02:     protected String boiled_pot;
03:     protected String boilwater(String elecpot, String water) {
04:         System.out.println(elecpot +"에 " + water +"을 넣어
    끓인다(Ramyon클래스)");
05:         return "끓은 물";
06:     }
07:     public void cookRamyon(String ramyon, String vegetable, int time) {
08:         boiled_pot = boilwater("전기 냄비","물");
09:         System.out.println(boiled_pot + "에 " + ramyon + "과 " + vegetable +
    "를 넣고 " + time + "분간 끓인다(Ramyon1클래스)");
10:         System.out.println("일반 라면 요리 완료(Ramyon1클래스)");
11:     }
12: }
13: class MixedRamyon extends Ramyon1 {
14:     public void cookRamyon(String ramyon, String vegetable, int time) {
15:         boiled_pot = boilwater("전기 냄비","물");
16:         System.out.println(boiled_pot + "에 " + ramyon + "과 " + vegetable +
    "을(를) 넣고 " + time + "분간 끓인다(MixedRamyon클래스)");
17:         System.out.println("면만 남기고 물을 따라 버린다(MixedRamyon클래스)");
18:         System.out.println("양념장을 넣고 비빈다(MixedRamyon클래스)");
19:         System.out.println("비빔라면 요리 완료(MixedRamyon클래스)");
```

메소드 오버라이딩(매개 변수의 형과 개수가 동일)

오버로딩한 메소드와 절차가 다르다

4-1 메소드 오버라이딩

● 예제 10.8

```
20:     }
21: }
22: public class OverridingTest2 {
23:     public static void main(String args[]) {
24:         int s;
25:         Ramyon1 r = new Ramyon1();
26:         r.cookRamyon("신라면", "파", 5);   상위 클래스의 메소드 호출
27:         MixedRamyon mr = new MixedRamyon();
28:         mr.cookRamyon("비빔면", "버섯", 6);   하위 클래스의 메소드 호
29:     }
30: }
```

실행 결과

```
전기냄비에 물을 넣어 끓인다(Ramyon1클래스)
끓은 물에 신라면과 파를 넣고 5분간 끓인다(Ramyon1클래스)
일반 라면 요리 완료(Ramyon1클래스)
전기 냄비에 물을 넣어 끓인다(Ramyon1클래스)
끓은 물에 비빔면과 버섯을(를) 넣고 6분간 끓인다(MixedRamyon클래스)
면만 남기고 물을 따라 버린다(MixedRamyon클래스)
양념장을 넣고 비빈다(MixedRamyon클래스)
비빔라면 요리 완료(MixedRamyon클래스)
```

4-2 메소드 오버라이딩과 한정자

- **메소드를 오버라이딩하는 경우 상위 클래스의 메소드 한정자보다 허용 범위가 넓은 경우에만 허용되고, 그 반대의 경우는 허용되지 않는다**
  - 상위 클래스에서 public으로 사용된 메소드를 하위 클래스에서 한정자 없이 사용하거나 protected로 메소드를 오버라이딩 할 수 없다.
  - 한정자의 허용 범위는 "public" → "protected" → "한정자 사용 안 함" 순서
  - private는 상속되지 않기 때문에 오버라이딩의 대상이 되지 않는다

4-2 메소드 오버라이딩과 한정자

한정자의 허용 범위가 넓어지는 경우는 허용됨

```
01: class AAA {
02:     void aa(int a, int b) {  <-- 한정자 없이 메소드 선언
03:         System.out.println(a+b);
04:     }
05: }
06: class BBB extends AAA {
07:     public void aa(int a, int b) {  <-- public 한정자 사용. 허용됨
08:         System.out.println(a*b);
09:     }
10: }
```

4-2 메소드 오버라이딩과 한정자

한정자의 허용 범위가 좁아지는 경우는 허용 안 됨. 컴파일 오류

```
01: class AAA {
02:     public void aa(int a, int b) {  <-- public 메소드 선언
03:         System.out.println(a+b);
04:     }
05: }
06: class BBB extends AAA {
07:     protected void aa(int a, int b) {  <-- protected 한정자 사용. 허용 안 됨
08:         System.out.println(a*b);
09:     }
10: }
```

4-2 메소드 오버라이딩과 한정자

private 한정자의 경우는 상속되지 않는다

```
01: class AAA {
02:     private void aa(int a, int b) {
03:         System.out.println(a+b);
04:     }
05: }
06: class BBB extends AAA {
07:     public void aa(int a, int b) {
08:         System.out.println(a*b);
09:     }
10: }
```

private 메소드 선언. 상속되지 않는다

상위 클래스와 상관없는 메소드. 같은 이름으로 정의 가능.

4-2 메소드 오버라이딩과 한정자

클래스 메소드의 경우에도 일반 메소드와 같은 규칙으로 오버라이딩이 가능하다

```
01: class AAA {
02:     static public void aa(int a, int b) {
03:         System.out.println(a+b);
04:     }
05: }
06: class BBB extends AAA {
07:     static void aa(int a, int b) {
08:         System.out.println(a*b);
09:     }
10: }
```

클래스 메소드 선언

클래스 메소드를 오버라이딩 하였으나,
한정자의 범위가 작기 때문에 오류 발생

4-2 메소드 오버라이딩과 한정자

클래스 메소드를 일반 메소드로 오버라이딩 할 수 없다

```
01: class AAA {
02:     static public void aa(int a, int b) {  <-------- 클래스 메소드 선언
03:         System.out.println(a+b);
04:     }
05: }
06: class BBB extends AAA {
07:     public void aa(int a, int b) {  <-------- 일반 메소드로 클래스 메소드를 오버라이딩. 오류 발생
08:         System.out.println(a*b);
09:     }
10: }
```

4-3 또 다른 주석 : @Override

- **메소드를 오버라이딩하여 사용할 때 유용하게 사용할 수 있는 주석**
  - @Override 주석 : 자바 컴파일러는 @Override 주석이 사용되어 메소드가 선언되면 상위 클래스의 메소드와 정확하게 일치하는지 검사하여 일치하지 않은 경우에는 컴파일 오류를 발생

오버라이딩되는 메소드의 이름을 잘못 기술한 경우

```
01: class OIa {
02:     public void show(String str) {
03:         System.out.println("상위 클래스의 메소드 show(String str)수행 "+str);
04:     }
05: }
06: class OIb extends OIa {
07:     public void shaw(String str) {   <-- 메소드 이름을 잘못 지정하여 오버라이딩.
08:         System.out.println("하위 클래스의 메소드 show(String str)수행 "+str);
09:     }
10: }
11: public class OverridingTest3 {
12:     public static void main(String args[]) {
13:         OIb oib = new OIb();
14:         oib.show("처음 시작하는 자바");   <-- 오버로딩된 메소드 show가 수행됨
15:     }
16: }
```

실행 결과

```
상위 클래스의 메소드 show(String str)수행 처음 시작하는 자바
```

4-3 또 다른 주석 : @Override

@Override 주석을 이용하여 오버라이딩되는 메소드를 기술한 경우

```
01: class OIa {
02:     public void show(String str) {
03:         System.out.println("상위 클래스의 메소드 show(String str)수행 "+str);
04:     }
05: }
06: class OIb extends OIa {
07:     @Override public void shaw(String str) {
08:         System.out.println("하위 클래스의 메소드 show(String str)수행 "+str);
09:     }
10: }
11: public class OverridingTest3 {
12:     public static void main(String args[]) {
13:         OIb oib = new OIb();
14:         oib.show("처음 시작하는 자바");
15:     }
16: }
```

@Override 주석을 이용하여 메소드를 선언. 상위 클래스의 오버라이딩 메소드와 일치하지 않아 오류를 발생시킨다

● 예약어 super는 두 가지 형태로 사용

- 첫 번째는 하위 클래스에 의해 가려진 상위 클래스의 멤버 변수나 메소드에 접근할 때 사용
- 두 번째는 상위 클래스의 명시적 생성자를 호출하기 위해 사용

[형식] super – 상위 클래스의 멤버 변수나 메소드에 접근

```
super.멤버 변수
super.메소드 이름(매개 변수)
```

[형식] super – 상위 클래스의 생성자 호출 : 반드시 첫 번째 라인에 위치해야 한다

```
super(매개 변수)
```

예제 10.9

예제 10.9 SuperTest1.java

```
01: class SB1 {
02:     public int x = 500;
03:     public int y = 1000;
04: }
05: class SB2 extends SB1 {
06:     public String x = "처음 시작하는 자바";   <-- 상위 클래스와 같은 이름의 멤버 변수 선언
07:     public String xx = x + super.x;
08:     public String yy = ""+ y + super.y;      <-- super를 사용하여 상위 클래스의 변수에 접근
09: }
10: public class SuperTest1 {
11:     public static void main(String args[]) {
12:         SB2 sb2 = new SB2();
13:         System.out.println("객체 sb2에 들어 있는 x,y 값 : " + sb2.x + sb2.y);
14:         //sb2.super.x 이와 같은 사용은 허용 안 됨
15:         System.out.println("객체 sb2에 들어 있는 xx 값 : " + sb2.xx);
16:         System.out.println("객체 sb2에 들어 있는 yy 값 : " + sb2.yy);
17:     }
18: }
```

실행 결과

```
객체 sb2에 들어 있는 x,y 값 : 처음 시작하는 자바1000
객체 sb2에 들어 있는 xx 값 : 처음 시작하는 자바500
객체 sb2에 들어 있는 yy 값 : 10001000
```

● 예제 10.10

```
01: class D1 {
02:     public int x = 1000;
03:     public void display() {
04:         System.out.println("상위 클래스 D1의 display() 메소드입니다");
05:     }
06: }
07: class D2 extends D1 {
08:     public int x = 2000;
09:     public void display() {
10:         System.out.println("하위 클래스 D2의 display() 메소드입니다");
11:     }
12:     public void write() {
13:         display();
14:         super.display();
15:         System.out.println("D2 클래스 객체의 x 값은 : " + x);
16:         System.out.println("D1 클래스 객체의 x 값은 : " + super.x);
17:     }
18: }
19: class SuperTest2 {
20:     public static void main(String args[]) {
21:         D2 d = new D2();
22:         d.write();
23:     }
24: }
```

상위 클래스와 같은 이름의 멤버 변수 선언

상위 클래스와 같은 이름의 메소드 선언

자신의 클래스에 있는 display() 호출

상위 클래스에 있는 display() 호출

자신의 x값을 출력

상위 클래스에 있는 x값 출력

실행 결과

```
하위 클래스 D2의 display() 메소드입니다
상위 클래스 D1의 display() 메소드입니다
D2 클래스 객체의 x 값은 : 2000
D1 클래스 객체의 x 값은 : 1000
```

● 예제 10.11

예제 10.11 SuperTest3.java

```
01: class SD1 {
02:     public int i1;
03:     public double d1;
04:     public SD1(int i1) {
05:         System.out.println("SD1(int i1) 생성자 수행");
06:         this.i1 = i1 * i1 ;  <-- this.i1은 객체 변수를 의미
07:         System.out.println(i1 +"의 2제곱은 : "+this.i1);
08:     }
09:     public SD1(double d1) {
10:         System.out.println("SD1(double d1) 생성자 수행");
11:         this.d1 = d1 * d1 ;
12:         System.out.println(d1 +"의 2제곱은 : "+this.d1);
13:    }
14: }
15: class Sub1 extends SD1 {
16:     public Sub1(int i1) {
17:         super(i1);  <-- 상위 클래스의 생성자 호출
18:         System.out.println("Sub1(int i1) 생성자 수행");
19:         this.i1 = this.i1 * i1 ;  <-- this.i1은 상속받은 객체 변수를 의미
20:         System.out.println(i1 +"의 3제곱은 : "+this.i1);
21:     }
```

● 예제 10.11

```
22:     public Sub1(double d1) {
23:         super(d1);
24:         System.out.println("Sub1(double d1) 생성자 수행");
25:         this.d1 = this.d1 * d1 ;
26:         System.out.println(d1 +"의 3제곱은 : "+this.d1);
27:     }
28: }
29: public class SuperTest3 {
30:     public static void main(String args[]) {
31:         Sub1 sub1 = new Sub1(10);
32:         Sub1 sub2 = new Sub1(10.5);
33:     }
34: }
```

상위 클래스의 생성자 호출

생성자에 의해 결과가 출력된다

실행 결과

```
SD1(int i1) 생성자 수행
10의 2제곱은 : 100
Sub1(int i1) 생성자 수행
10의 3제곱은 : 1000
SD1(double d1) 생성자 수행
10.5의 2제곱은 : 110.25
Sub1(double d1) 생성자 수행
10.5의 3제곱은 : 1157.625
```

- Object 클래스는 java.lang 패키지에 속해 있는 라이브러리 클래스
- 모든 자바 클래스의 최상위 클래스는 Object 클래스
- 명시적으로 상위 클래스가 지정된 경우에도, 그 상위 클래스의 상위 클래스가(계속 반복 가능) 최종적으로는 Object 클래스
  - 즉 Object 클래스에서 선언된 모든 속성과 기능은 모든 자바 클래스에 상속되기 때문에 자유롭게 사용할 수 있다

6 Object 클래스

표 10-2 Object 클래스의 메소드

메소드	설명
protected Object clone()	객체의 복사본을 만들어 Object 형의 객체로 반환
public boolean equals (Object object)	현재의 객체와 object로 지정된 객체가 같으면 true, 다르면 false를 반환
protected void finalize()	자바에서는 객체가 더 이상 사용되지 않으면 자동적으로 쓰레기 수집(garbage collection) 기능을 수행한다. finalize() 메소드는 쓰레기 수집 기능이 수행되기 전에 호출되며 객체가 점유하고 있던 자원들을 해제하는 데 사용된다.
public Class<?> getClass()	객체의 클래스명을 Class형의 객체로 반환
public int hashCode()	호출한 객체와 연관된 해쉬 hash 코드를 얻는다. 해쉬 코드는 메모리에 저장된 객체의 16진수 주소를 의미한다.
public String toString()	현재 객체의 문자열 표현을 반환
public void notify()	대기 중인 스레드 중 하나의 스레드를 다시 시작시킨다.
public void notifyAll()	대기 중인 모든 스레드를 다시 시작시킨다.
public void wait()	스레드의 실행을 중지하고 대기 상태로 간다.
public void wait (long millisec)	스레드의 실행을 중지하고 millisec 밀리초 동안 대기한 다음 다시 시작한다.
public void wait (long millisec, int nanosec)	스레드의 실행을 중지하고 millisec 밀리초 + nanosec 나노초 동안 대기한 다음 다시 시작한다.

6-1 toString() 메소드

- **Object 클래스에 선언된 toString() 메소드는 객체의 클래스명과 메모리에서의 주소를 16진수로 반환하는 메소드**

Object 클래스에 선언된 toString() 메소드

```
01: public String toString() {
02:     return getClass().getName() +        <-- 현재 클래스의 이름을 반환
03:             "@" + Integer.toHexString(hashCode());
04: }
```

hashCode()의 값은 객체가 저장된 메모리의 10진 주소이다. 이 값을 16진수로 변환하여 반환. Integer 클래스의 toHexString()은 클래스 메소드로서 10진수를 16진수로 변환하는 메소드이다.

Integer 클래스는 라이브러리 클래스로서 정수를 저장할 수 있는 클래스입니다. 다양한 메소드(클래스 메소드)를 제공하고 있습니다. Integer 클래스에 대해서는 해당 부분에서 설명합니다.

6-1 toString() 메소드

● 예제 10.12

예제 10.12 ObjectMethodTest1.java

```
01: class AAA {
02:     public int a;
03: }
04: public class ObjectMethodTest1 {
05:     public static void main(String args[]) {
06:     AAA aa = new AAA();
07:     System.out.println(aa);
08:     System.out.println(aa.toString());
09:     Integer ii = new Integer(99);
10:     System.out.println(ii);
11:     System.out.println(ii.toString());
12:     }
13: }
```

07: 객체 aa의 toString() 메소드가 자동으로 호출 (Object 클래스로부터 상속되었음)

08: 객체 aa의 toString() 메소드를 명시적으로 호출. 결과가 같음.

09: Integer 클래스로부터 객체 ii를 생성. 객체의 값은 99

10: 객체 ii의 toString() 메소드가 자동으로 호출.

11: 명시적으로 toString() 메소드를 호출. 결과가 같음.

실행 결과

```
AAA@15bdc50
AAA@15bdc50
99
99
```

6-1 toString() 메소드

● 예제 10.13

예제 10.13 ObjectMethodTest2.java

```
01: class AAA1 {
02:     public int a;
03:     public String toString() {          <-- Object 클래스의 toString() 메소드 오버라이딩
04:         return "AAA1 클래스 객체, 속성 a의 값은 : " + a;   <-- 객체의 특성을 나타냄
05:     }
06: }
07: public class ObjectMethodTest2 {
08:     public static void main(String args[]) {
09:     AAA1 aa = new AAA1();
10:     System.out.println(aa);
11:     System.out.println(aa.toString());   <-- toString()에서 지정된 객체의 특성을 출력함
12:     }
13: }
```

실행 결과

```
AAA1 클래스 객체, 속성 a의 값은 : 0
AAA1 클래스 객체, 속성 a의 값은 : 0
```

6-2 equals() 메소드

- **앞에서 두 개의 변수값을 비교하는 연산자로 동등 연산자(==)를 사용**
- **동등 연산자를 참조 자료형의 변수에 적용할 수는 있지만, 예상치 않은 결과를 나타내게 된다.**
  - 기본 자료형 변수는 가지고 있는 값이 실제 값인 반면에, 참조 자료형 변수는 가지고 있는 값이 실제 값이 아니라 객체가 저장된 주소이기 때문

6-2 equals() 메소드

- 예제 10.14

예제 10.14 ObjectMethodTest3.java

```
01: class Box10 {
02:     public int width;
03:     public int height;
04:     public int depth;
05:     public Box10(int w,int h,int d) {
06:         width = w;
07:         height = h;
08:         depth = d;
09:     }
10: }
11: public class ObjectMethodTest3 {
12:     public static void main(String args[]) {
13:         int a1=10, a2=10;
14:         Box10 b1 = new Box10(10,20,30);
15:         Box10 b2 = new Box10(10,20,30);
16:         Box10 b3 = b2;
17:         System.out.println(a1==a2 ? "a1과 a2는 같다" :"a1과 a2는 같지 않다" );
18:         System.out.println(b1==b2 ? "b1과 b2는 같다" :"b1과 b2는 같지 않다" );
19:         System.out.println(b2==b3 ? "b2와 b3는 같다" :"b2와 b3는 같지 않다" );
20:     }
21: }
```

기본 자료형 변수에 같은 값을 배정

같은 속성값을 가진 Box 클래스 객체 생성

Box 객체 b3에 b2를 배정

동등 연산자를 이용한 비교

● 예제 10.14

실행 결과

```
a1과 a2는 같다
b1과 b2는 같지 않다
b2와 b3는 같다
```

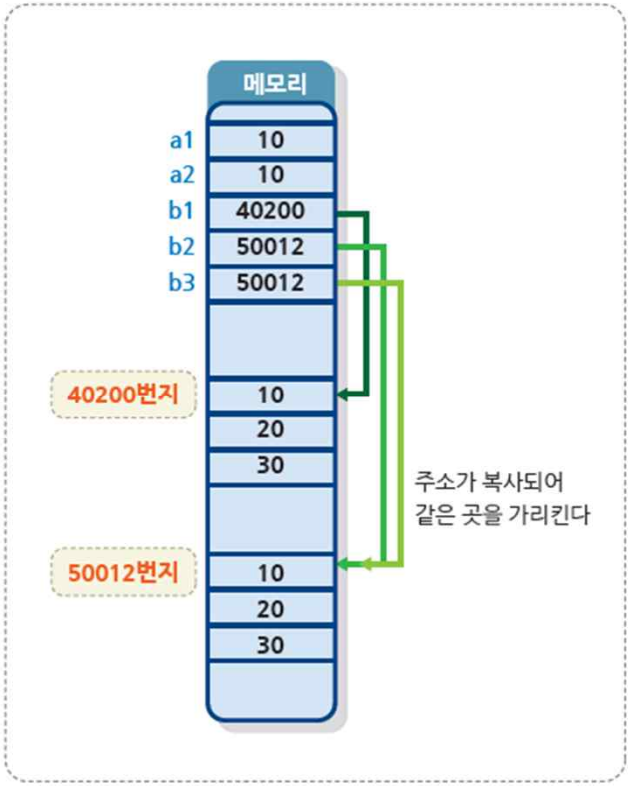


그림 10-4 예제 10.13의 메모리 구조

6-2 equals() 메소드

Object 클래스에 선언된 equals() 메소드

```
01: public boolean equals(Object obj) {
02:     return (this == obj);
03: }
```

두 객체 변수의 값(주소)이 같으면 true, 아니면 false를 반환

6-2 equals() 메소드

- 예제 10.15

예제 10.15 ObjectMethodTest4.java

```
01: class Box11 {
02:     public int width;
03:     public int height;
04:     public int depth;
05:     public Box11(int w,int h,int d) {
06:         width = w;
07:         height = h;
08:         depth = d;
09:     }
10: }
11: public class ObjectMethodTest4 {
12:     public static void main(String args[]) {
13:         Box11 b1 = new Box11(10,20,30);
14:         Box11 b2 = new Box11(10,20,30);
15:         Box11 b3 = b2;
16:             System.out.println(b1.equals(b2) ? "b1과 b2는 같다  :"b1과 b2는
    같지 않다" );
17:             System.out.println(b2.equals(b3) ? "b2와 b3는 같다" :"b2와 b3는
    같지 않다" );
18:             String s1 = new String("처음 시작하는 자바");
19:             String s2 = new String("처음 시작하는 자바");
20:             System.out.println(s1.equals(s2) ? "s1과 s2는 같다" :"s1과 s2는
    같지 않다" );
21:     }
22: }
```

equals() 메소드를 이용하여 객체를 비교(동등 연산자와 같은 결과)

String 클래스의 객체를 이용하여 비교(String 클래스에 오버라이딩 된 equals() 수행)

실행 결과

```
b1과 b2는 같지 않다
b2와 b3는 같다
s1과 s2는 같다
```

6-2 equals() 메소드

- 예제 10.16

```
01: class Box12 {
02:     public int width;
03:     public int height;
04:     public int depth;
05:     public Box12(int w,int h,int d) {
06:         width = w;
07:         height = h;
08:         depth = d;
09:     }
10:     public boolean equals(Box12 box12) {   <-- equals() 메소드를 오버라이딩. 객체의 속성을 비교
11:         return (this.width == box12.width && this.height == box12.height &&
    this.depth == box12.depth);
12:     }
13: }
14: public class ObjectMethodTest5 {
15:     public static void main(String args[]) {
16:         Box12 b1 = new Box12(10,20,30);
17:         Box12 b2 = new Box12(10,20,30);
18:         Box12 b3 = b2;                                   논리적, 물리적으로 같은지를 비교
19:         System.out.println(b1.equals(b2) ? "b1과 b2는 논리적으로 같다" :"b1과
    b2는 논리적으로 같지 않다" );
20:             System.out.println(b1 == b2 ? "b1과 b2는 물리적으로 같다" :"b1과
    b2는 물리적으로 같지 않다" );
21:         System.out.println(b2.equals(b3) ? "b2와 b3는 논리적으로 같다" :"b2와
    b3는 논리적으로 같지 않다" );
22:         System.out.println(b2 == b3 ? "b2와 b3는 물리적으로 같다" :"b2와 b3는
    물리적으로 같지 않다" );
23:     }
24: }
```

실행 결과

```
b1과 b2는 논리적으로 같다
b1과 b2는 물리적으로 같지 않다
b2와 b3는 논리적으로 같다
b2와 b3는 물리적으로 같다
```

- **자바에서 final은 3가지 형태로 사용 : 모두 변하는 것을 방지하기 위해 사용**
  - 첫 번째는 메소드 지역 변수나 객체 변수에 final을 붙여 상수로 사용
  - final의 두 번째 용도는 메소드에 final을 붙여 선언하는 경우. 이 경우에는 하위 클래스에서 이 메소드를 오버라이딩 할 수 없다는 의미
  - 마지막으로 final은 클래스 선언에도 사용할 수 있다. final이 붙은 클래스는 상속을 허용하지 않는다

7 예약어 final

- 첫 번째는 메소드 지역 변수나 객체 변수에 final을 붙여 상수로 사용

```
01: class AAA {
02:     public final int MAX = 100;  <-- 객체 변수에 final을 사용하여 상수로 지정
03:     ................
04:     public void temp() {
05:         final int max = 50;  <-- 메소드 지역 변수를 final로 지정
06:         .....
07:         max++;  <-- 오류 발생. final로 지정된 지역 변수의 값을 변경할 수 없음
08:     }
09: }
10: class BBB {
11:     ................
12:      AAA aa = new AAA();
13:     aa.MAX = 200;  <-- 오류 발생. final로 지정된 객체 변수의 값을 변경할 수 없음
14:     ................
15: }
```

7 예약어 final

- final의 두 번째 용도는 메소드에 final을 붙여 선언하는 경우. 이 경우에는 하위 클래스에서 이 메소드를 오버라이딩 할 수 없다는 의미

```
01: class AAA {
02:     final void calculate() {  <-- 메소드를 final로 선언
03:         System.out.println("이 메소드는 final 메소드");
04:     }
05: }
06: class BBB extends AAAA {
07:     void calculate()  {  <-- 오류 발생. final로 선언된 메소드 오버라이딩 금지
08:         System.out.println("final 메소드를 중첩하는 메소드");
09:     }
10: }
```

7 예약어 final

- 마지막으로 final은 클래스 선언에도 사용할 수 있다. final이 붙은 클래스는 상속을 허용하지 않는다

```
01: final class AAA {          <-- 클래스를 final로 선언
02:     ..........
03: }
04: class BBB extends AAA {    <-- 오류 발생 final로 선언된 클래스 상속 금지
05:     .........
06: }
```

학습 정리

- **상속Inheritance의 개요**

  ① 절차 지향에서의 모듈의 재사용이 가능하기는 하지만, 비슷한 모듈의 양산과 코드의 중복 문제가 발생하게 됩니다.

  ② 상속은 객체 지향의 주요 특성인 모듈의 재사용과 코드의 간결성을 제공합니다.

  ③ 객체 지향에서 상속은 확장의 개념으로 상위 클래스의 모든 요소를 상속받고 추가 요소를 더 가지는 개념입니다.

  ④ 상속은 다양한 효과와 특성을 가집니다.

- **상속과 한정자**

  ① 클래스가 상속되면 상위 클래스에 선언된 멤버 변수는 접근 한정자에 따라 상속 여부가 결정됩니다.

  ② protected 접근 한정자는 같은 패키지 내의 클래스와 같은 패키지는 아니지만, 상속된 클래스에서 사용 가능한 접근 한정자입니다.

학습 정리

- **상속과 생성자**

  ① 상속 관계에서 매개 변수가 없는 생성자(묵시적 생성자)는 하위 클래스에서 객체가 생성될 때 자동으로 수행됩니다.

  ② 묵시적 생성자가 아닌 경우에는 명시적으로 super를 사용하여 호출하여야 합니다.

- **상속과 메소드 오버라이딩**

  ① 클래스가 상속되면 상위 클래스에 선언된 메소드도 접근 한정자에 따라 상속 여부가 결정됩니다.

  ② 상위 클래스에서 선언된 메소드와 같은 메소드를 하위 클래스에 선언하는 것을 오버라이딩이라 합니다.

  ③ 오버라이딩이 성립되기 위해서는 상위 클래스의 메소드와 매개 변수의 형과 개수가 정확하게 일치해야 합니다.

  ④ 상속 관계에서 메소드가 오버라이딩되면 상위 클래스의 메소드는 가려지게 됩니다.

  ⑤ 오버로딩의 개념이 중첩이라면, 오버라이딩의 개념은 상위 클래스의 메소드를 대치하는치환에 해당됩니다.

  ⑥ 메소드가 오버라이딩 될 때 상위 클래스 메소드의 한정자보다는 접근 허용 범위가 넓어야 합니다.

학습 정리

- **예약어 super**

  ① super는 하위 클래스에 의해 가려진 상위 클래스의 멤버 변수나 메소드에 접근할 때 사용합니다.

  ② 상위 클래스의 생성자를 호출할 때도 사용합니다.

- **Object 클래스**

  ① 모든 자바 클래스의 최상위 클래스는 Object 클래스입니다.

  ② 모든 자바 프로그램에는 Object 클래스에서 제공되는 메소드가 상속되므로 사용할 수 있습니다.

  ③ Object 클래스에서 제공되는 toString() 메소드는 객체의 특성을 나타내기 위해 사용되는 메소드로서 출력문의 매개 변수로 객체가 사용될 때 자동으로 호출되는 메소드입니다.

  ④ Object 클래스에서 제공되는 equals() 메소드는 객체의 동등 관계를 나타내는 메소드로서 물리적으로 같은 장소에 있는지를 비교하여 결과를 반환합니다.

  ⑤ toString() 메소드와 equals() 메소드를 클래스 작성 시 오버라이딩하여 유용하게 사용할 수 있습니다.

학습 정리

- 예약어 final
  - ① 객체 변수나 메소드 지역 변수를 final로 선언하여 변하지 않는 상수값을 지정할 수 있습니다.
  - ② 메소드를 final로 지정하면 하위 클래스에서 오버라이딩 할 수 없습니다.
  - ③ 클래스를 final로 지정하면 하위 클래스를 가질 수 없다는 의미입니다.
  - ④ final을 사용하는 이유는 보안과 설계의 명확화를 위해서입니다.