

Week 3

안드로이드 프로그래밍



한성대학교 컴퓨터공학부
한 기 준 교 수

안드로이드를 위한 JAVA 문법



Java 의 역사

- ✓ 1991년 선마이크로시스템스(오라클에 인수됨)의 제임스 고슬링이 C 언어를 모델로 연구 시작
- ✓ 1995년 JDK(Java Development Kit) 1.0 발표
- ✓ 1997년 JDK 1.1이 발표되면서 완전한 프로그래밍 언어의 모습을 갖추

Java의 특징

- ① 구문이 간결함
- ② 명료한 객체지향 언어
- ③ 이식성이 높고, 기계에 종립적
- ④ 분산 처리 지원
- ⑤ 멀티스레드(Multi-thread) 언어



Java 프로그램 전통적인 작성법

- ✓ 메모장에서 Java 코드를 작성한 후에 *.java로 저장
 - ✓ javac.exe를 사용해서 컴파일하면 *.class 파일이 생성
 - ✓ java.exe를 사용해서 컴파일된 *.class 파일을 실행
- 개발자들은 대부분 이클립스 환경에서 Java 개발



■ 변수 선언 예제

예제 3-2 exam02.java

```
1 public class exam02 {  
2     public static void main(String args[]) {  
3         int var1 = 10;  
4         float var2 = 10.1f;  
5         double var3 = 10.2;  
6         char var4 = '안';  
7         String var5 = "안드로이드";  
8         System.out.println(var1);  
9         System.out.println(var2);  
10        System.out.println(var3);  
11        System.out.println(var4);  
12        System.out.println(var5);  
13    }  
14 }
```

```
10  
10.1  
10.2  
안  
안드로이드
```



Java에서 많이 사용되는 기본적인 데이터 형식

표 3-1 Java에서 주로 사용되는 데이터 형식

데이터 형식		설명
문자형	char	2byte를 사용하며 한글 또는 영문 1개만 입력
	String	여러 글자의 문자열을 입력
정수형	byte	1byte를 사용하며 -128~+127까지 입력
	short	2byte를 사용하며 -32768~+32767까지 입력
	int	4byte를 사용하며 약 -21억~+21억까지 입력
	long	8byte를 사용하며 상당히 큰 정수까지 입력 가능
실수형	float	4byte를 사용하며 실수를 입력
	double	8byte를 사용하며 실수를 입력. float보다 정밀도가 높음
불리언형	boolean	true 또는 false를 입력



if문

- ✓ 조건이 true, false인지에 따라서 어떤 작업을 할 것인지를 결정

```
if(조건식) {  
    // 조건식이 true일 때 이 부분 실행  
}
```

```
if(조건식) {  
    // 조건식이 true일 때 이 부분 실행  
} else {  
    // 조건식이 false일 때 이 부분 실행  
}
```



switch()~case문

- ✓ 여러 가지 경우에 따라 어떤 작업을 할 것인지를 결정

```
switch( 값 ) {  
  case 값1 :  
    // 값1이면 이 부분 실행  
    break;  
  case 값2 :  
    // 값2이면 이 부분 실행  
    break;  
  :  
  default :  
    // 아무것도 해당하지 않으면 이 부분 실행  
    break;  
}
```


조건문 if, switch() ~ case



예제 3-3 exam03.java

```
1 public class exam03 {
2     public static void main(String args[]) {
3         int count = 85;
4         if (count >= 90) {
5             System.out.println("if문: 합격 (장학생)");
6         } else if (count >= 60) {
7             System.out.println("if문: 합격");
8         } else {
9             System.out.println("if문: 불합격");
10        }
11
12        int jumsu = (count / 10) * 10;
13        switch (jumsu) {
14            case 100:
15            case 90:
16                System.out.println("switch문: 합격(장학생)");
17                break;
18            case 80:
19            case 70:
20            case 60:
21                System.out.println("switch문: 합격");
22                break;
23            default:
24                System.out.println("switch문: 불합격");
25        }
26    }
27 }
```

if문: 합격
switch문: 합격



배열

- ✓ 여러 데이터를 한 변수에 저장하는 데 사용

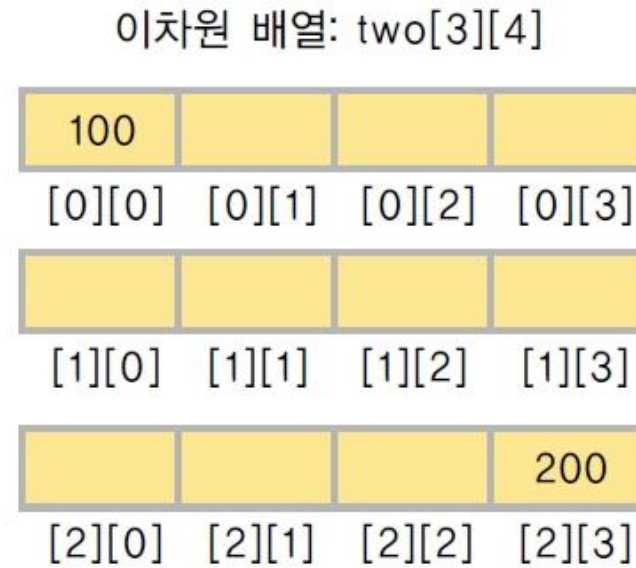
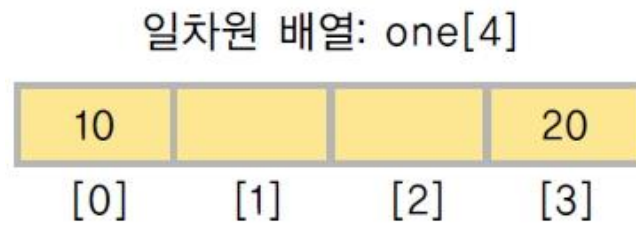


그림 3-5 배열의 개념



일차원 배열

```
int one[] = new int[4];  
one[0] = 10;  
one[3] = 20;
```

이차원 배열

```
int two[][] = new int[3][4];  
two[0][0] = 100;  
two[2][3] = 200;
```



배열 선언하면서 바로 값 대입하기

```
int three[] = { 1, 2, 3 };
```

배열 크기 확인하기

- ✓ '배열.length' 사용



for문

- ✓ 조건문과 함께 프로그래밍의 필수 요소임

```
for(초기식; 조건식; 증감식) {  
    // 이 부분을 반복 실행  
}
```

배열을 지원하는 for문의 형식

- ✓ 배열의 내용이 하나씩 변수에 대입된 후 for문 내부가 실행됨
- ✓ 결국 배열의 개수만큼 for문이 반복됨

```
for(변수형 변수 : 배열명) {  
    // 이 부분에서 변수를 사용  
}
```



while문

```
while( 조건식 ) {  
    // 조건식이 true인 동안 이 부분을 수행  
}
```

반복문: for, while



예제 3-4 exam04.java

```
1 public class exam04 {  
2     public static void main(String args[]) {  
3         int one[] = new int[3];  
4         for (int i = 0; i < one.length; i++) {  
5             one[i] = 10 * i;  
6         }  
7  
8         String two[] = { "하나", "둘", "셋" };  
9         for (String str : two) {  
10            System.out.println(str);  
11        }  
12  
13        int j=0;  
14        while( j < one.length ) {  
15            System.out.println(one[j]);  
16            j++;  
17        }  
18    }  
19 }
```

하나
둘
셋
0
10
20



try~catch

- ✓ 프로그램 실행 중에 발생하는 오류를 Java는 try~catch문을 통해 처리

예제 3-6 exam06.java

```
1 public class exam06 {  
2     static int var = 100;  
3     public static void main(String args[]) {  
4         int num1 = 100, num2 = 0;  
5         try {  
6             System.out.println(num1/num2);  
7         }  
8         catch (java.lang.ArithmeticException e) {  
9             System.out.println("계산에 문제가 있습니다.");  
10        }  
11    }  
12 }
```

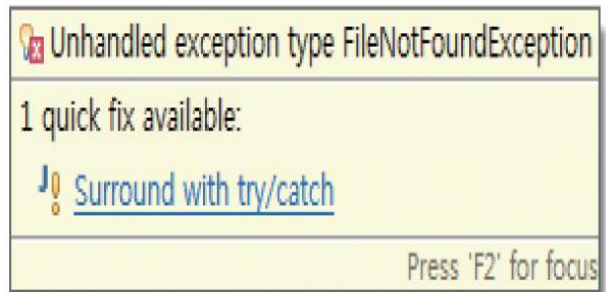
계산에 문제가 있습니다.



■ 안드로이드에서의 try~catch

- ✓ 안드로이드에서는 예외 처리에 자동 완성을 많이 사용함
- ✓ 'Surround with try/catch' 클릭

```
FileOutputStream outFs = openFileOutput("file.txt",2);
```



```
try {  
    FileOutputStream outFs = openFileOutput("file.txt",2);  
} catch (FileNotFoundException e) {  
    // TODO Auto-generated catch block  
    e.printStackTrace();  
}
```



■ JAVA에서 주로 사용되는 연산자

표 3-2 주로 사용되는 Java 연산자

연산자	설명
+, -, *, /, %	사칙 연산자로 %는 나머지값을 계산한다.
+, -	부호 연산자로 변수, 수, 식 앞에 붙일 수 있다.
=	대입 연산자로 오른쪽을 왼쪽에 대입한다.
++, --	1씩 증가 또는 감소시킨다.
==, !=, <, >, >=, <=	비교 연산자로 결과는 true 또는 false이며, if문이나 반복문의 조건식에 주로 사용된다.
&&,	논리 연산자로 and, or를 의미한다.
&, , ^, ~	비트 연산자로, 비트 단위로 and, or, exclusive or, not 연산을 한다.
<<, >>	시프트 연산자로, 비트 단위로 왼쪽 또는 오른쪽으로 이동한다.
+=, -=, *=, /=	복합 대입 연산자로, 'a+=b'는 'a=a+b'와 동일하다.
(데이터 형식)	캐스트(cast) 연산자로, 데이터 형식을 강제로 변환한다. 예를 들어 int a = (int) 3.5는 double형인 3.5 값을 int형으로 강제로 변환하여 a에 대입한다. 결국 a에 3이 대입된다.



■ 캐스트 연산자

- ✓ 안드로이드 프로그래밍에서 클래스형 데이터의 강제 형식 변환에도 상당히 많이 사용됨
- ✓ 안드로이드 프로그래밍에서 캐스트 연산자 사용 예
 - View 클래스형을 Button형으로 변환

```
Button button1;  
button1 = (Button) findViewById(R.id.btn1);
```

객체지향 프로그래밍(Object-Oriented Programming, OOP)

✓ Java, C++, C# 등에서 사용되는 프로그래밍 기술

클래스(class) = 변수(필드) + 메소드

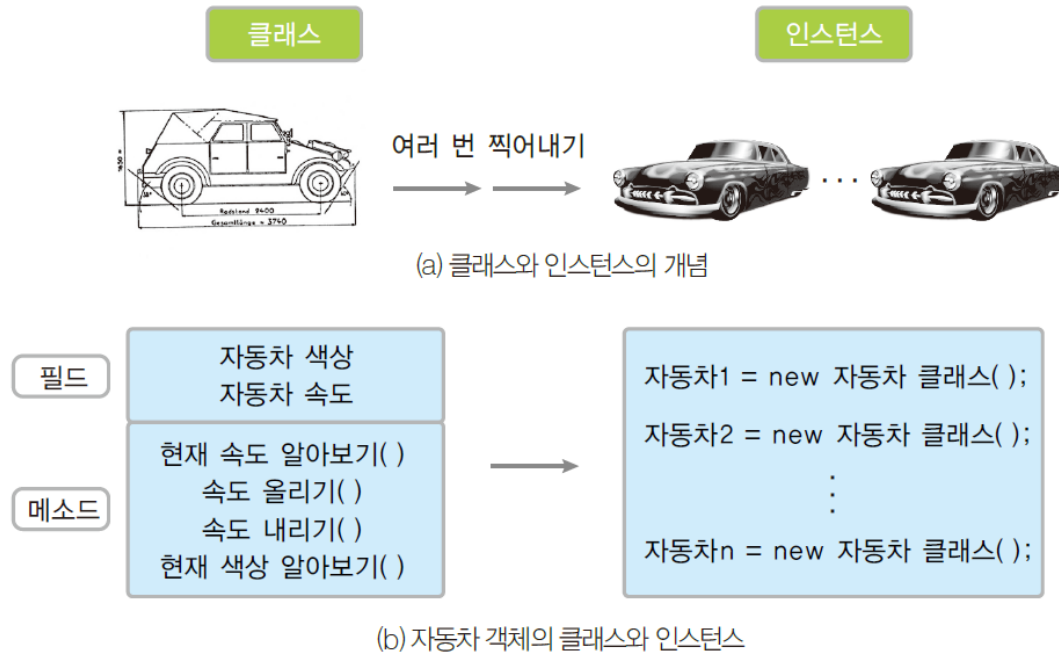


그림 3-6 클래스의 형식



자동차 클래스의 코드 구현

예제 3-7 Car.java-기본 구조

```
1 public class Car {
2     String color;
3     int speed = 0;
4
5     int getSpeed() {
6         return speed;
7     }
8
9     void upSpeed(int value) {
10        if (speed + value >= 200)
11            speed = 200;
12        else
13            speed = speed + value;
14    }
15
16    void downSpeed(int value) {
17        if (speed - value <= 0)
18            speed = 0;
19        else
20            speed = speed - value;
21    }
22
23    String getColor() {
24        return color;
25    }
26 }
```



자동차 인스턴스 생성

예제 3-8 exam07.java

```
1 public class exam07 {
2     public static void main(String args[]) {
3         Car myCar1 = new Car();
4         myCar1.color = "빨강";
5         myCar1.speed = 0;
6
7         Car myCar2 = new Car();
8         myCar2.color = "파랑";
9         myCar2.speed = 0;
10
11        Car myCar3 = new Car();
12        myCar3.color = "초록";
13        myCar3.speed = 0;
14
15        myCar1.upSpeed(50);
16        System.out.println("자동차1의 색상은 " + myCar1.getColor()
17            + "이며, 속도는 "
18            + myCar1.getSpeed() + "km입니다.");
19
20        myCar2.downSpeed(20);
21        ~~~~ 중간 생략(myCar2 내용 출력) ~~~~
22        myCar3.upSpeed(250);
23        ~~~~ 중간 생략(myCar3 내용 출력) ~~~~
24    }
25 }
```

자동차1의 색상은 빨강이며, 속도는 50km입니다.
자동차2의 색상은 파랑이며, 속도는 0km입니다.
자동차3의 색상은 초록이며, 속도는 200km입니다.



생성자로 인스턴스 만들기

- ✓ [예제 3-7]의 Car.java에 생성자 코드를 추가

예제 3-9 Car.java-생성자 추가

```
1 public class Car {
2     String color;
3     int speed;
4
5     Car(String color, int speed) {
6         this.color = color;
7         this.speed = speed;
8     }
9     ~~~~ 중간 생략([예제 3-7]의 5행 이하와 동일) ~~~~
```

- ✓ [예제 3-8]의 Car 클래스를 사용한 myCar1, myCar2, myCar3의 내용을 변경

예제 3-10 exam07.java-수정

```
1 public class exam07 {
2     public static void main(String args[]) {
3         Car myCar1 = new Car("빨강",0);
4         Car myCar2 = new Car("파랑",0);
5         Car myCar3 = new Car("초록",0);
6
7         ~~~~ 중간 생략([예제 3-8]의 15행 이하와 동일) ~~~~
```



메소드 오버로딩

- ✓ 클래스 내에서 메소드의 이름이 같아도 파라미터의 개수나 데이터형만 다르면 여러 개 선언 가능

예제 3-11 Car.java-메소드 오버로딩 추가

```
1 public class Car {  
2     String color;  
3     int speed;  
4  
5     Car(String color, int speed) {  
6         this.color = color;  
7         this.speed = speed;  
8     }  
9  
10    Car(int speed) {  
11        this.speed = speed;  
12    }  
13  
14    Car() {  
15    }  
16  
17    ~~~~ 중간 생략([예제 3-7]의 5행 이하와 동일) ~~~~
```




정적 필드(static field)

- ✓ 클래스 자체에서 사용되는 변수

정적 메소드(static method)

- ✓ 메소드 앞에 static 붙여 사용
- ✓ 인스턴스 없이 '클래스명.메소드명()'으로 호출해서 사용

상수 필드

- ✓ 정적 필드에 초기값을 입력하고 final을 앞에 붙임



예제 3-12 Car.java-정적 구성 요소 추가

```
1 public class Car {
2     String color;
3     int speed;
4     static int carCount = 0;
5     final static int MAXSPEED = 200;
6     final static int MINSPEED = 0;
7
8     static int currentCarCount() {
9         return carCount;
10    }
11
12    Car(String color, int speed) {
13        this.color = color;
14        this.speed = speed;
15        carCount ++;
16    }
17
18    ~~~~ 중간 생략([예제 3-11]의 10행 이하와 동일) ~~~~
```



정적 구성 요소 추가

예제 3-13 exam08.java

```
1  import java.lang.Math;
2
3  public class exam08 {
4      public static void main(String args[]) {
5          Car myCar1 = new Car("빨강", 0);
6          Car myCar2 = new Car("파랑", 0);
7          Car myCar3 = new Car("초록", 0);
8
9          System.out.println("생산된 차의 대수(정적 필드) ==> " + Car.carCount);
10         System.out.println("생산된 차의 대수(정적 메소드) ==> " + Car.currentCarCount());
11         System.out.println("차의 최고 제한 속도 ==> " + Car.MAXSPEED);
12
13         System.out.println("PI의 값 ==> " + Math.PI);
14         System.out.println("3의 5제곱 ==> " + Math.pow(3, 5));
15     }
16 }
```

생산된 차의 대수(정적 필드) ==> 3
생산된 차의 대수(정적 메소드) ==> 3
차의 최고 제한 속도 ==> 200
PI의 값 ==> 3.141592653589793
3의 5제곱 ==> 243.0



클래스 상속(inheritance)

- ✓ 기존 클래스를 그대로 물려받으면서 필요한 필드나 메소드를 추가로 정의
- ✓ 슈퍼클래스(super class, 또는 부모 클래스) → 자동차 클래스
- ✓ 서브클래스(subclass, 또는 자식 클래스) → 승용차 클래스와 트럭 클래스

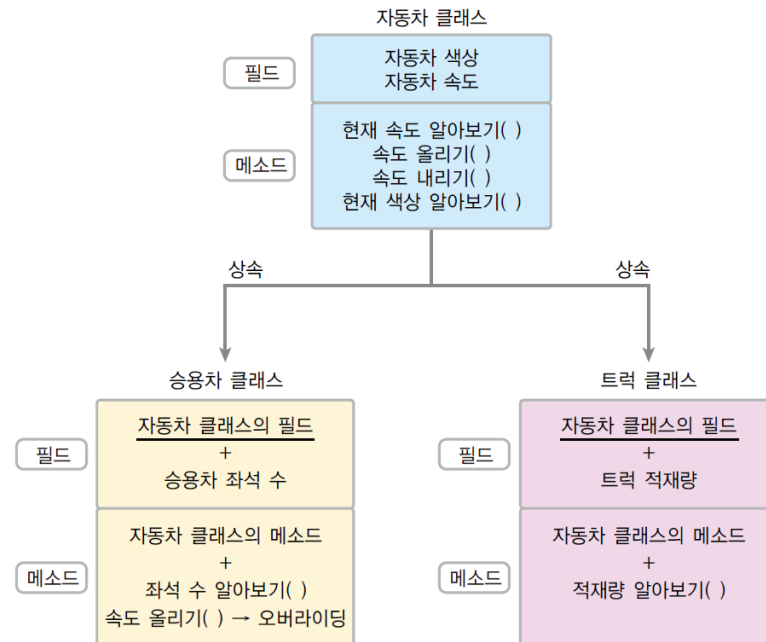


그림 3-7 클래스 상속의 개념



승용차 클래스를 Java 코드로 변경한 예제

예제 3-14 Automobile.java

```
1 public class Automobile extends Car {  
2     int seatNum;  
3  
4     int getSeatNum() {  
5         return seatNum;  
6     }  
7  
8     void upSpeed(int value) {  
9         if (speed + value >= 300)  
10            speed = 300;  
11        else  
12            speed = speed + (int) value;  
13    }  
14 }
```



서브클래스를 Java 코드로 변경한 예제

예제 3-15 exam09.java

```
1 public class exam09 {  
2     public static void main(String args[]) {  
3         Automobile auto = new Automobile();  
4  
5         auto.upSpeed(250);  
6         System.out.println("승용차의 속도는 "  
7             + auto.getSpeed() + "km입니다.");  
8     }  
9 }
```

승용차의 속도는 250km입니다.



■ 추상(abstract) 클래스

- ✓ 인스턴스화를 금지하는 클래스
- ✓ 메소드 앞에 abstract 써서 사용

■ 추상 메소드

- ✓ 메소드 본체가 없는 메소드
- ✓ 메소드 앞에 abstract 써서 사용
- ✓ 추상 메소드를 포함하는 클래스는 추상 클래스로 지정해야 함
- ✓ 추상 메소드를 오버라이딩하는 것을 추상 메소드를 '구현한다(implement)'고 함



❖ 동물 클래스를 추상 클래스로 만들고 추상 메소드인 '이동한다()'를 포함하는 도식

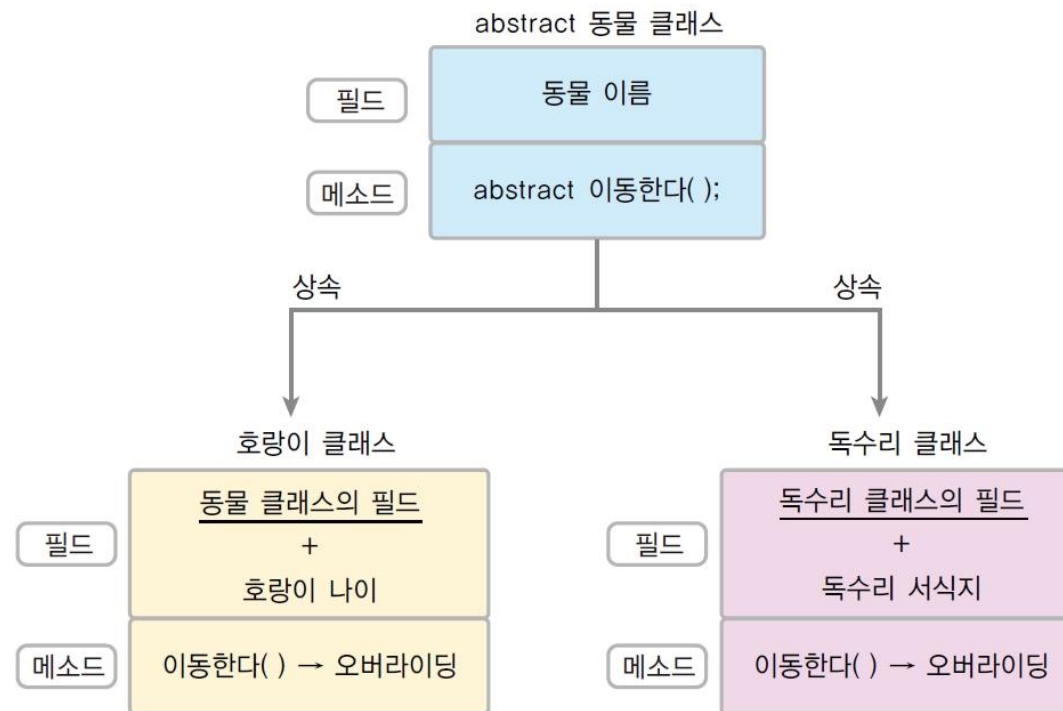


그림 3-8 추상 클래스 상속의 개념



■ [그림 3-8]을 코드로 구현하기

예제 3-16 Animal.java

```
1 abstract class Animal {  
2     String name;  
3     abstract void move();  
4 }
```

예제 3-17 Tiger.java

```
1 class Tiger extends Animal {  
2     int age;  
3     void move() {  
4         System.out.println("네발로 이동한다.");  
5     }  
6 }
```

예제 3-18 Eagle.java

```
1 class Eagle extends Animal {  
2     String home;  
3     void move() {  
4         System.out.println("날개로 이동한다.");  
5     }  
6 }
```



■ [그림 3-8]을 코드로 구현하기

예제 3-19 exam10.java

```
1 public class exam10 {  
2     public static void main(String args[]) {  
3         Tiger tiger1 = new Tiger();  
4         Eagle eagle1 = new Eagle();  
5  
6         tiger1.move();  
7         eagle1.move();  
8     }  
9 }
```

네발로 이동한다.
날개로 이동한다.



다형성(polymorphism)

- ✓ 자신의 서브 클래스에서 생성한 인스턴스도 클래스 변수에 대입할 수 있는 것

예제 3-20 exam11.java

```
1 public class exam11 {  
2     public static void main(String args[]) {  
3         Animal animal;  
4  
5         animal = new Tiger();  
6         animal.move();  
7  
8         animal = new Eagle();  
9         animal.move();  
10    }  
11 }
```

네발로 이동한다.
날개로 이동한다.



익명 내부 클래스(Anonymous inner class)

- ✓ '이름이 없는' 내부 클래스
- ✓ 한 번만 사용하고 버려지는 클래스에 사용

예제 3-22 exam13.java

```
1 interface clickListener {
2     public void print();
3 }
4
5 public class exam13 {
6     public static void main(String args[]) {
7
8         clickListener listener =
9             (new clickListener() {
10                 public void print() {
11                     System.out.println("클릭 리스너입니다.");
12                 }
13             });
14
15         listener.print();
16     }
17 }
```

클릭 리스너입니다.



패키지(Package)

- ✓ 클래스와 인터페이스가 많아지면 관리가 어려워 패키지 단위로 묶어서 관리
- ✓ [New]-[Package]를 선택
- ✓ 사용자가 생성한 클래스가 포함될 패키지는 *.java 파일 맨 첫 행에 지정

package 패키지명;

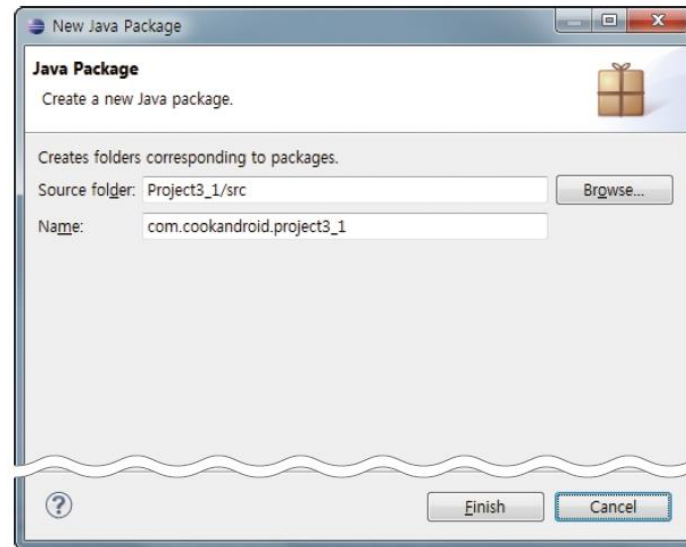


그림 3-9 패키지 생성



■ 제네릭스(Generics)

- ✓ 데이터 형식의 안전성을 보장하는 데 사용
- ✓ <String>뿐 아니라 <Integer>, <Double>, 사용자가 정의한 클래스형에 사용

```
ArrayList strList = new ArrayList();  
strList.add("첫 번째");  
strList.add("두 번째");  
strList.add(3);
```



```
ArrayList<String> strList = new ArrayList<String>();  
strList.add("첫 번째");  
strList.add("두 번째");  
strList.add(3);
```

데이터 형식 변환

- ✓ 데이터형 변환을 위해 캐스팅 연산자 대신 Java에서 제공하는 클래스의 정적 메소드 사용

```
int a = Integer.parseInt("100");  
double b = Double.parseDouble("100.123");
```

문자열 비교

- ✓ 문자열을 비교하려면 String 클래스의 equals() 메소드를 사용

```
String str = "안녕하세요";  
if (str.equals((String)"안녕하세요" )) {  
    // 문자열이 같으면 이곳을 수행  
}
```

■ 날짜 형식

- ✓ 날짜를 표현하기 위해 DateFormat 클래스를 사용
- ✓ 이를 상속받은 SimpleDateFormat을 사용하면 '연월일'이나 '시분초'와 같은 표현이 가능

```
Date now = new Date();
SimpleDateFormat sFormat;

sFormat = new SimpleDateFormat("yyyyMMdd");
System.out.println(sFormat.format(now)); // 20121131 형식으로 출력

sFormat = new SimpleDateFormat("HH:mm:ss");
System.out.println(sFormat.format(now)); // 23:15:21 형식으로 출력
```