

안드로이드를 위한 Java 문법

학습목표

- 기본적인 Java 문법을 익힌다.
- 안드로이드 프로그래밍을 위한 Java의 특징을 이해한다.

목차

01 Java의 개요

02 Java의 기본 문법

03 클래스와 인스턴스

04 클래스 상속

05 추가로 알아둘 Java 문법

5. 메소드와 전역변수, 지역변수

■ 변수

- 전역변수(global variable) : 모든 메소드에서 사용 가능함
- 지역변수(local variable) : 메소드 내부에서만 사용 가능함

예제 3-5 exam05.java

```
1 public class exam05 {  
2     static int var = 100;  
3     public static void main(String args[]) {  
4         int var = 0;  
5         System.out.println(var);  
6  
7         int sum = addFunction(10, 20);  
8         System.out.println(sum);  
9     }  
10  
11     static int addFunction(int num1, int num2) {  
12         int hap;  
13         hap = num1 + num2 + var;  
14         return hap;  
15     }  
16 }
```

0
130

6. 예외 처리: try~catch

■ 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.

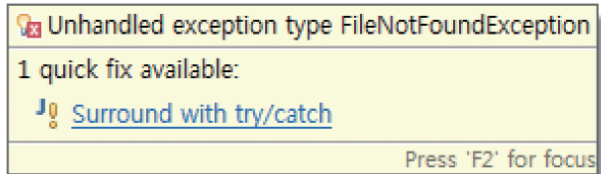
6. 예외 처리: try~catch

저자
한마디

try~catch 자동 완성

안드로이드에서는 예외 처리에 자동 완성을 많이 사용한다. 아래 그림에서 밑줄이 그어진 `openFileOutput()`에 마우스를 가져가면 팝업 창이 나오는데, 이때 'Surround with try/catch'를 클릭하면 된다.

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



try~catch문이 자동 완성된 결과는 다음과 같다. 자세한 내용은 8장에서 살펴보겠다.

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

7. 연산자

■ 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.

7. 연산자

■ 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);
```


1. 클래스 정의와 인스턴스 생성

■ 객체지향 프로그래밍(Object-Oriented Programming, OOP)

- Java, C++, C# 등에서 사용되는 프로그래밍 기술

■ 클래스(class) = 변수(필드) + 메소드

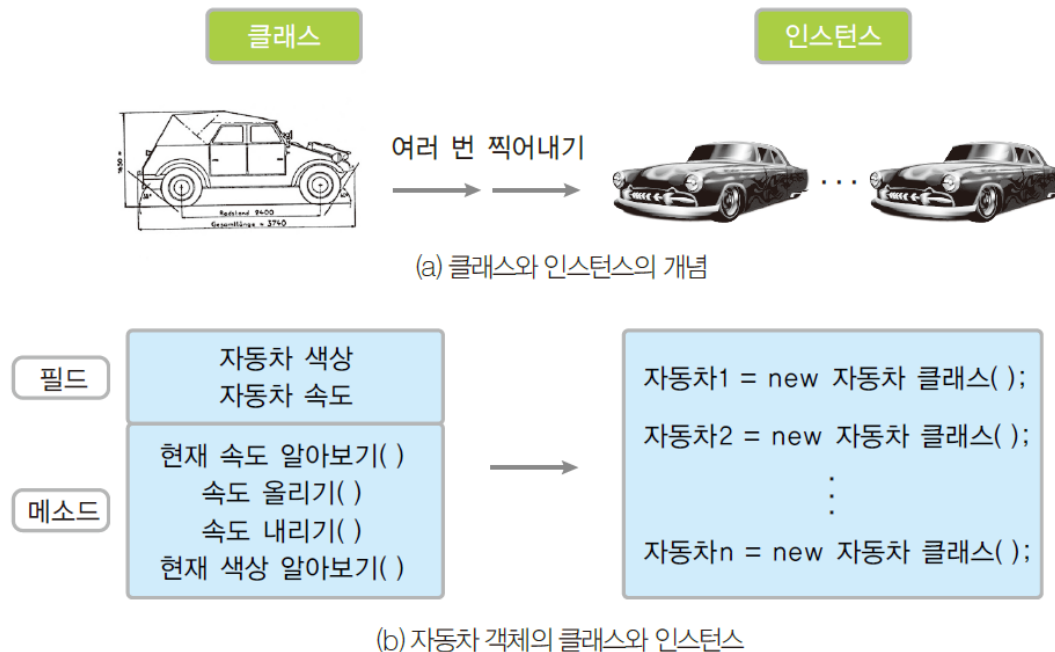


그림 3-6 클래스의 형식

1. 클래스 정의와 인스턴스 생성

■ 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 }
```

1. 클래스 정의와 인스턴스 생성

■ Car 클래스를 인스턴스로 생성하기

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.

2. 생성자

■ 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) ~~~~
```

3. 메소드 오버로딩

■ 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) ~~~~
```

4. 정적 필드, 정적 메소드, 상수 필드

■ 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

4. 정적 필드, 정적 메소드, 상수 필드

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) ~~~~
```

4. 정적 필드, 정적 메소드, 상수 필드

■ 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

1. 클래스 상속과 메소드 오버라이딩

■ 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

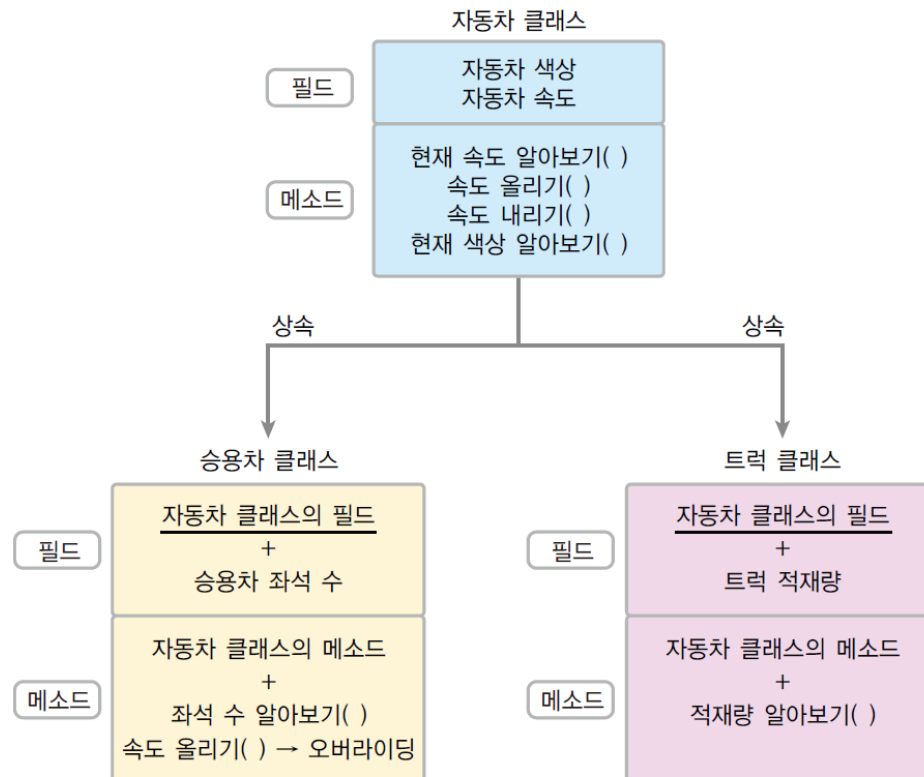


그림 3-7 클래스 상속의 개념

1. 클래스 상속과 메소드 오버라이딩

■ 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 }
```

1. 클래스 상속과 메소드 오버라이딩

■ 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.

2. 추상 클래스와 추상 메소드

■ 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

2. 추상 클래스와 추상 메소드

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

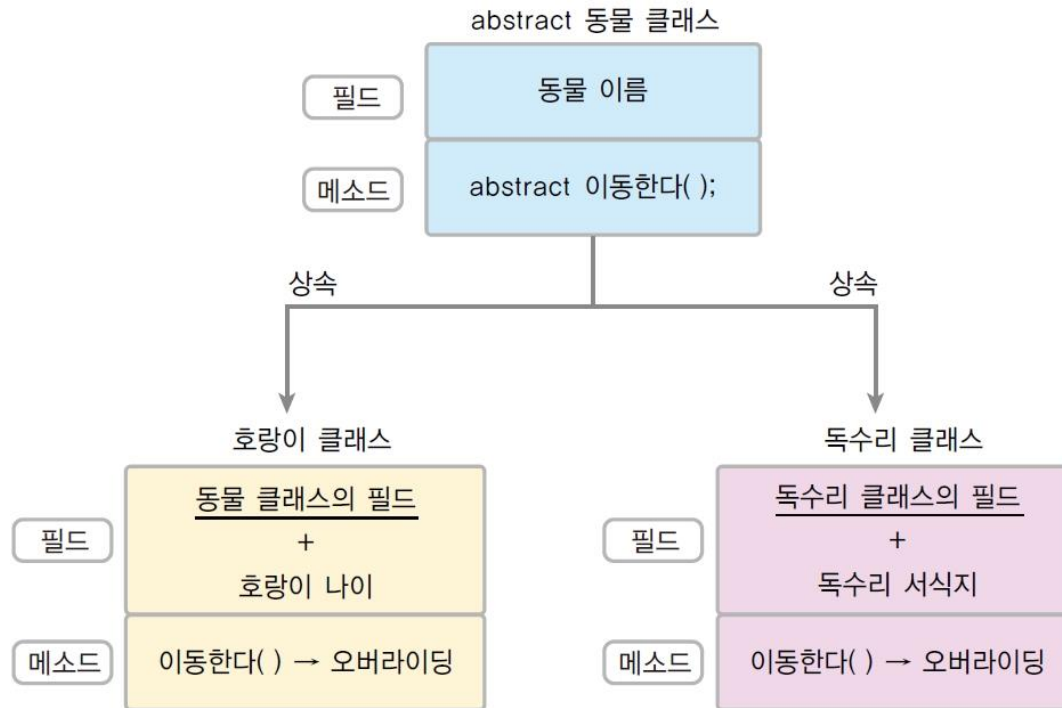


그림 3-8 추상 클래스 상속의 개념

2. 추상 클래스와 추상 메소드

■ [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 }
```

2. 추상 클래스와 추상 메소드

■ [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.

3. 클래스 변수의 다형성

■ 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.

4. 인터페이스와 다중 상속

■ 인터페이스(Interface)

- class 키워드 대신 interface 키워드를 사용해서 정의
- 내부에는 추상 메소드를 선언
- 클래스에서 인터페이스를 받아서 완성할 때 implements 키워드 사용

4. 인터페이스와 다중 상속

■ 다중 상속

- Java는 다중 상속을 지원하지 않음. 대신 인터페이스를 사용하여 비슷하게 작성 가능함

예제 3-21 exam12.java

```
1 interface iAnimal {
2     abstract void eat();
3 }
4
5 public class exam12 {
6     public static void main(String args[]) {
7         iCat cat = new iCat();
8         cat.eat();
9
10        iTiger tiger = new iTiger();
11        tiger.move();
12        tiger.eat();
13    }
14
15    static class iCat implements iAnimal {
```

생선을 좋아한다.
네발로 이동한다.
멧돼지를 잡아먹는다.

```
16        public void eat() {
17            System.out.println("생선을 좋아한다.");
18        }
19    }
20
21    static class iTiger extends Animal implements iAnimal {
22        void move() {
23            System.out.println("네발로 이동한다.");
24        }
25
26        public void eat() {
27            System.out.println("멧돼지를 잡아먹는다.");
28        }
29    }
30 }
```

5. 익명 내부 클래스

■ 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.

1. 패키지

■ 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;
```

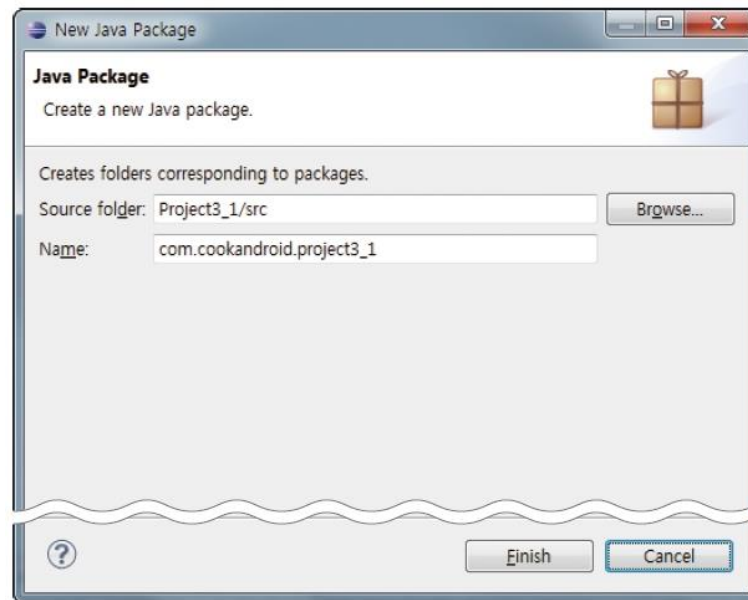


그림 3-9 패키지 생성

2. 제네릭스

■ 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);
```

3. 데이터 형식 변환, 문자열 비교, 날짜 형식

■ 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  
}
```

3. 데이터 형식 변환, 문자열 비교, 날짜 형식

■ 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
```