이항연산자와 삼항 연산자

이항연산자

❖ 이항 연산자란?

- 피연산자가 2개인 연산자

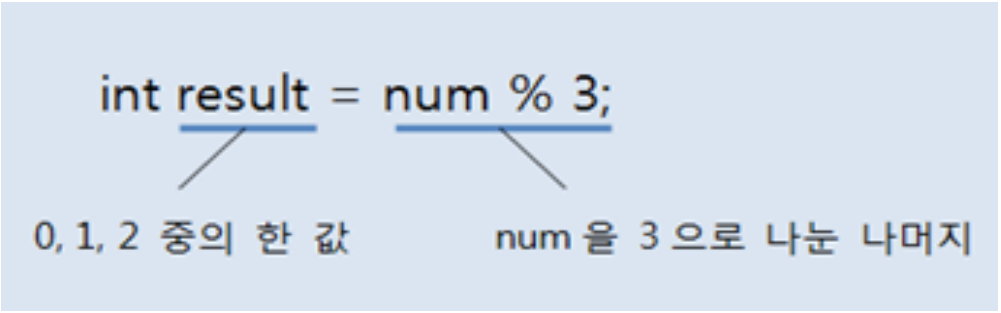
- 종류
  - 산술 연산자: +, -, *, /, %
  - 문자열 연결 연산자: +
  - 대입 연산자: =, +=, -=, *=, /=, %=, &=, ^=, |=, <<=, >>=, >>>=
  - 비교 연산자: <, <=, >, >=, ==, !=
  - 논리 연산자: &&, ||, &, |, ^, !
  - 비트 논리 연산자: &, |, ^
  - 비트 이동 연산자: <<, >>, >>>

이항연산자

❖ 산술 연산자

- boolean 타입을 제외한 모든 기본 타입에 사용 가능
- 결과값 산출할 때 Overflow 주의
- 정확한 계산은 정수를 사용
- NaN과 Infinity 연산은 주의할 것

연산식			설명
피연산자	+	피연산자	덧셈 연산
피연산자	-	피연산자	뺄셈 연산
피연산자	*	피연산자	곱셈 연산
피연산자	/	피연산자	좌측 피연산자를 우측 피연산자로 나눗셈 연산
피연산자	%	피연산자	좌측 피연산자를 우측 피연산자로 나눈 나머지를 구하는 연산



이항연산자

❖ 산술 연산자: ArithmeticOperatorExample.java

```
public class ArithmeticOperatorExample {
   public static void main(String[] args) {
      int v1 = 5;
      int v2 = 2;

      int result1 = v1 + v2;
      System.out.println("result1=" + result1);
      int result2 = v1 - v2;
      System.out.println("result2=" + result2);

      int result3 = v1 * v2;
      System.out.println("result3=" + result3);
      int result4 = v1 / v2;
      System.out.println("result4=" + result4);

      int result5 = v1 % v2;
      System.out.println("result5=" + result5);

      double result6 = (double) v1 / v2;
      System.out.println("result6=" + result6);
   }
}
```

이항연산자

❖ 산술 연산자: CharOperationExample.java

```
public class CharOperationExample {
   public static void main(String[] args) {
      char c1 = 'A' + 1;
      char c2 = 'A';
      // char c3 = c2 + 1; //컴파일 에러

      System.out.println("c1: " + c1);
      System.out.println("c2: " + c2);
      // System.out.println("c3: " + c3);
   }
}
```

❖ 오버플로우 탐지: OverflowExample.java

```
public class OverflowExample {
   public static void main(String[] args) {
      int x = 1000000;
      int y = 1000000;
      int z = x * y;
      System.out.println(z);

      int x = 1000000;
      int y = 1000000;
      double z = (double)x * y;
      System.out.println(z);

      /*
      long x = 1000000;
      long y = 1000000;
      long z = x * y;
      System.out.println(z);
      */
   }
}
```

❖ 문자열 연산자: StringConcatExample.java

```
public class StringConcatExample {
   public static void main(String[] args) {
      String str1 = "JDK" + 6.0;
      String str2 = str1 + " 특징";
      System.out.println(str2);

      String str3 = "JDK" + 3 + 3.0;
      String str4 = 3 + 3.0 + "JDK";
      System.out.println(str3);
      System.out.println(str4);
   }
}
```

이항연산자

❖ 비교 연산자(==, !=, <, >, <=, >=) (p.87~91)

- 대소(<, <=, >, >=) 또는 동등(==, !=) 비교해 boolean 타입인 true/false 산출

구분	연산식			설명
동등 비교	피연산자	==	피연산자	두 피 연산자의 값이 같은지를 검사
	피연산자	!=	피연산자	두 피 연산자의 값이 다른지를 검사
크기 비교	피연산자	>	피연산자	피 연산자 1 이 큰지를 검사
	피연산자	>=	피연산자	피 연산자 1 이 크거나 같은지를 검사
	피연산자	<	피연산자	피 연산자 1 이 작은지를 검사
	피연산자	<=	피연산자	피 연산자 1 이 작거나 같은지를 검사

- 동등 비교 연산자는 모든 타입에 사용
- 크기 비교 연산자는 boolean 타입 제외한 모든 기본 타입에 사용
- 흐름 제어문인 조건문(if), 반복문(for, while)에서 주로 이용
- 실행 흐름을 제어할 때 사용

❖ 비교 연산자: StringConcatExample.java

```
public class CompareOperatorExample1 {
   public static void main(String[] args) {
      int num1 = 10;
      int num2 = 10;

      boolean result1 = (num1 == num2);
      boolean result2 = (num1 != num2);
      boolean result3 = (num1 <= num2);

      System.out.println("result1=" + result1);
      System.out.println("result2=" + result2);
      System.out.println("result3=" + result3);

      char char1 = 'A';
      char char2 = 'B';
      boolean result4 = (char1 < char2);
      System.out.println("result4=" + result4);
   }
}
```

❖ 비교 연산자: StringConcatExample.java

```
public class CompareOperatorExample2 {
   public static void main(String[] args) {
      int v2 = 1;
      double v3 = 1.0;
      System.out.println(v2 == v3); // true

      double v4 = 0.1;
      float v5 = 0.1f;
      System.out.println(v4 == v5); // false
      System.out.println((float) v4 == v5); // true
      System.out.println((int) (v4 * 10) == (int) (v5 * 10)); // true
   }
}
```

이항연산자

❖ 논리 연산자 (&&, ||, &, |, ^, !)

- 논리곱(&&), 논리합(||), 배타적 논리합(^) ,논리 부정(!) 연산 수행
- 피연산자는 boolean 타입만 사용 가능

구분	연산식			결과	설명
AND (논리곱)	true	&& 또는 &	true	true	피 연사자 모두가 true 일 경우에만 연산 결과는 true
	true		false	false	
	false		true	false	
	false		false	false	
OR (논리합)	true	|| 또는 |	true	true	피 연산자 중 하나만 true 이면 연산 결과는 true
	true		false	true	
	false		true	ture	
	false		false	false	
XOR (배타적 논리합)	true	^	true	false	피 연산자가 하나는 ture 이고 다른 하나가 false 일 경우에만 연산 결과는 true
	true		false	true	
	false		true	ture	
	false		false	false	
NOT (논리부정)		!	true	false	피 연산자의 논리값을 바꿈
			false	true	

❖ 논리 연산자: LogicalOperatorExample.java

```
public class LogicalOperatorExample {

   public static void main(String[] args) {

      int charCode = 'A';

      if ((charCode >= 65) & (charCode <= 90)) {
         System.out.println("대문자 이군요");
      }

      if ((charCode >= 97) && (charCode <= 122)) {
         System.out.println("소문자 이군요");
      }

      if (!(charCode < 48) && !(charCode > 57)) {
         System.out.println("0~9 숫자 이군요");
      }
```

❖ 논리 연산자: LogicalOperatorExample.java

```
        int value = 6;

        if ((value % 2 == 0) | (value % 3 == 0)) {
            System.out.println("2 또는 3의 배수 이군요");
        }

        if ((value % 2 == 0) || (value % 3 == 0)) {
            System.out.println("2 또는 3의 배수 이군요");
        }
    }
}
```

이항연산자

❖ 비트 연산자(&, |, ^, ~, <<, >>, >>>)

- 비트(bit) 단위로 연산 하므로 0과 1이 피연산자
  - 0과 1로 표현이 가능한 정수 타입만 비트 연산 가능
  - 실수 타입인 float과 double은 비트 연산 불가

- 종류
  - 비트 논리 연산자(&, |, ^, ~)
  - 비트 이동 연산자(<<, >>, >>>)

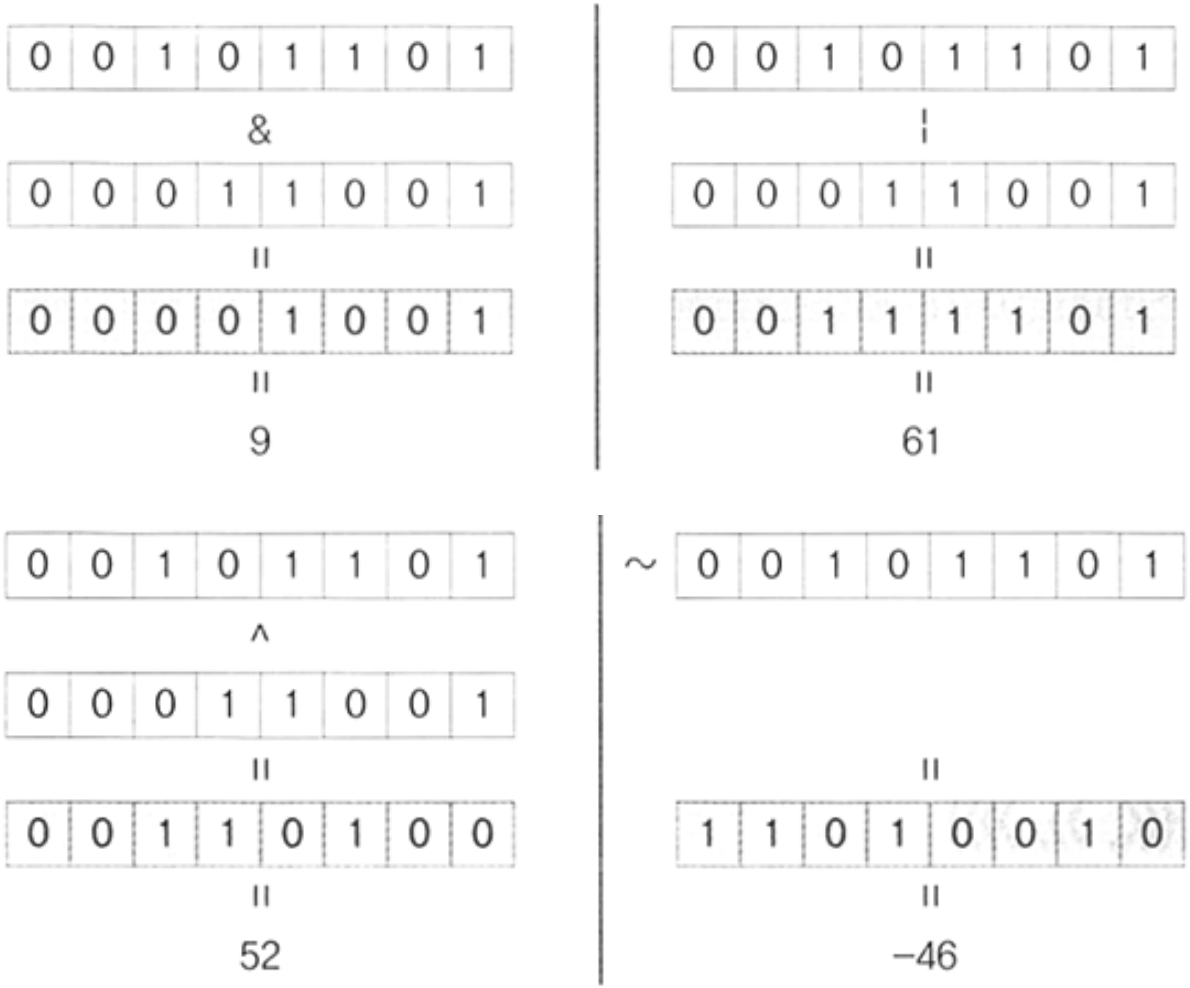
이항연산자

❖ 비트 논리 연산자(&, |, ^, ~)

- 피 연산자가 boolean타입일 경우 일반 논리 연산자
- 피연산자가 정수 타입일 경우 비트 논리 연산자로 사용

구분	연산식			결과	설명
AND (논리곱)	1	&	1	1	두 비트가 모두가 1 일 경우에만 연산 결과는 1
	1		0	0	
	0		1	0	
	0		0	0	
OR (논리합)	1	|	1	1	두 비트 중 하나만 1 이면 연산 결과는 1
	1		0	1	
	0		1	1	
	0		0	0	
XOR (배타적 논리합)	1	^	1	0	두 비트 중 하나는 1 이고 다른 하나가 0 일 경우 연산 결과는 1
	1		0	1	
	0		1	1	
	0		0	0	
NOT (논리부정)		~	1	0	보수
			0	1	

❖ 비트 논리 연산자(&, |, ^, ~)


0 0 1 0 1 1 0 1
&
0 0 0 1 1 0 0 1
=
0 0 0 0 1 0 0 1
=
9
0 0 1 0 1 1 0 1
|
0 0 0 1 1 0 0 1
=
0 0 1 1 1 1 0 1
=
61
0 0 1 0 1 1 0 1
^
0 0 0 1 1 0 0 1
=
0 0 1 1 0 1 0 0
=
52
~ 0 0 1 0 1 1 0 1
=
1 1 0 1 0 0 1 0
=
−46

이항연산자

❖ 비트 논리 연산자(&, |, ^, ~)

- 비트 연산자는 피연산자를 int타입으로 자동 타입 변환 후 연산 수행

```
byte num1 = 45;
byte num2 = 25;
byte result = num1 & num2; //컴파일 에러    ➡    int result = num1 & num2;
```

❖ 비트 논리 연산자: BitLogicExample.java

```
public class BitLogicExample {
   public static void main(String[] args) {
      System.out.println("45 & 25 = " + (45 & 25));
      System.out.println("45 | 25 = " + (45 | 25));
      System.out.println("45 ^ 25 = " + (45 ^ 25));
      System.out.println("~45 = " + (~45));

      System.out.println(toBinaryString(45));
      System.out.println("&");
      System.out.println(toBinaryString(25));
      System.out.println("||");
      System.out.println(toBinaryString(45 & 25));
   }

   public static String toBinaryString(int value) {
      String str = Integer.toBinaryString(value);
      while (str.length() < 32) {
         str = "0" + str;
      }
      return str;
   }
}
```

이항연산자

❖ 비트 이동 연산자(<<, >>, >>>)

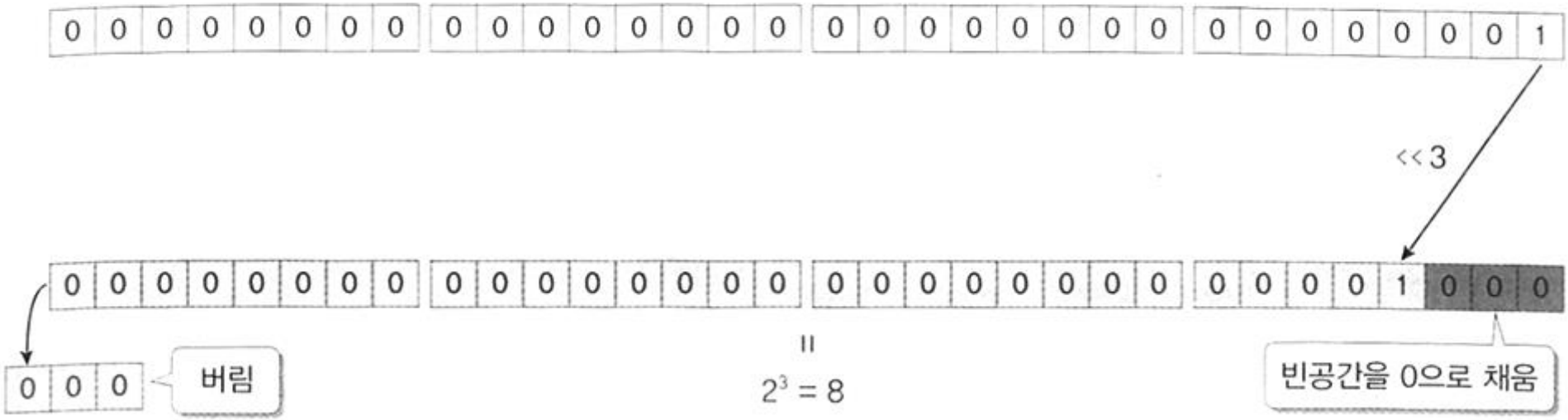
- 정수 데이터의 비트를 좌측 또는 우측으로 밀어 이동시키는 연산 수행

구분	연산식			설명
이동 (쉬프트)	a	<<	b	정수 a 의 각 비트를 b 만큼 왼쪽으로 이동 (빈자리는 0 으로 채워진다.)
	a	>>	b	정수 a 의 각 비트를 b 만큼 오른쪽으로 이동 (빈자리는 정수 a 의 최상위 부호 비트(MSB)와 같은 값으로 채워진다.)
	a	>>>	b	정수 a 의 각 비트를 오른쪽으로 이동 (빈자리는 0 으로 채워진다.)

이항연산자

❖ 비트 이동 연산자(<<, >>, >>>)

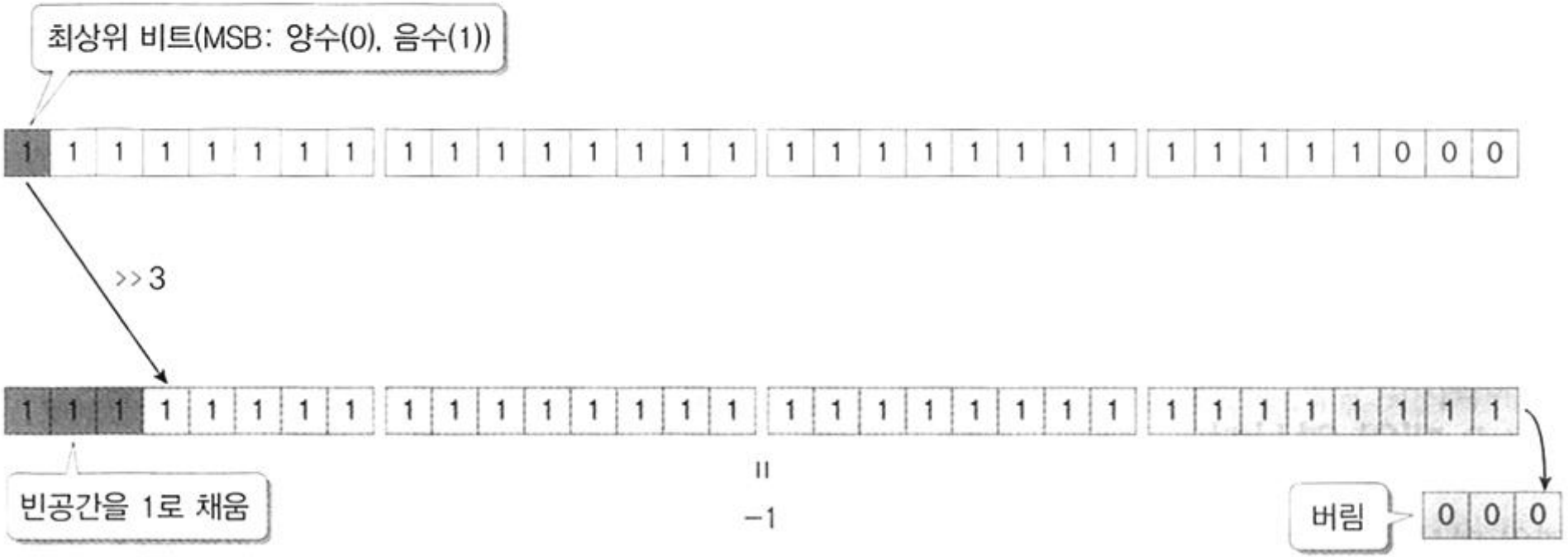
```
int result = 1 << 3;
```



이항연산자

❖ 비트 이동 연산자(<<, >>, >>>)

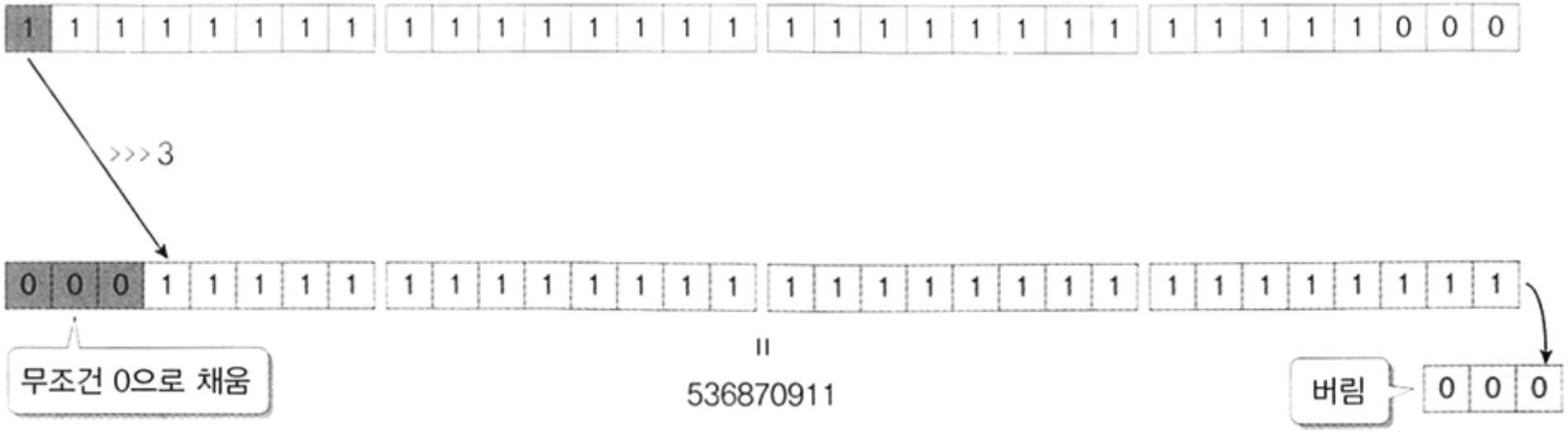
```
int result = -8 >> 3;
```



이항연산자

❖ 비트 이동 연산자(<<, >>, >>>)

```
int result = -8 >>>3;
```



❖ 비트 이동 연산자: BitShiftExample.java

```
public class BitShiftExample {
   public static void main(String[] args) {
      System.out.println("1 << 3 = " + (1 << 3));
      System.out.println("-8 >> 3 = " + (-8 >> 3));
      System.out.println("-8 >>> 3 = " + (-8 >>> 3));

      System.out.println(toBinaryString(1));
      System.out.println("<< 3");
      System.out.println(toBinaryString(1 << 3));
   }

   public static String toBinaryString(int value) {
      String str = Integer.toBinaryString(value);
      while (str.length() < 32) {
         str = "0" + str;
      }
      return str;
   }
}
```

이항연산자

❖ **대입 연산자(=, +=, -=, *=, /=, %=, &=, ^=, |=, <<=, >>=, >>>=)**

- 오른쪽 피연산자의 값을 좌측 피연산자인 변수에 저장

- `result += 10;  →  result = result  + 10;`

- 모든 연산자들 중 가장 낮은 연산 순위 → 제일 마지막에 수행

- 종류
  - 단순 대입 연산자
  - 복합 대입 연산자
  - 정해진 연산을 수행한 후 결과를 변수에 저장

이항연산자

❖ 대입 연산자의 종류

구분	연산식			설명
단순 대입 연산자	변수	=	피연산자	우측의 피연산자의 값을 변수에 저장
복합 대입 연산자	변수	+=	피연산자	우측의 피연산자의 값을 변수의 값과 더한 후에 다시 변수에 저장 (변수=변수+피연산자 와 동일)
	변수	-=	피연산자	우측의 피연산자의 값을 변수의 값에서 뺀 후에 다시 변수에 저장 (변수=변수-피연산자 와 동일)
	변수	*=	피연산자	우측의 피연산자의 값을 변수의 값과 곱한 후에 다시 변수에 저장 (변수=변수*피연산자 와 동일)
	변수	/=	피연산자	우측의 피연산자의 값으로 변수의 값을 나눈 후에 다시 변수에 저장 (변수=변수/피연산자 와 동일)
	변수	%=	피연산자	우측의 피연산자의 값으로 변수의 값을 나눈 후에 나머지를 변수에 저장 (변수=변수%피연산자 와 동일)
	변수	&=	피연산자	우측의 피연산자의 값과 변수의 값을 & 연산 후 결과를 변수에 저장 (변수=변수&피연산자 와 동일)
	변수	|=	피연산자	우측의 피연산자의 값과 변수의 값을 | 연산 후 결과를 변수에 저장 (변수=변수|피연산자 와 동일)
	변수	^=	피연산자	우측의 피연산자의 값과 변수의 값을 ^ 연산 후 결과를 변수에 저장 (변수=변수^피연산자 와 동일)
	변수	<<=	피연산자	우측의 피연산자의 값과 변수의 값을 << 연산 후 결과를 변수에 저장 (변수=변수<<피연산자 와 동일)
	변수	>>=	피연산자	우측의 피연산자의 값과 변수의 값을 >> 연산 후 결과를 변수에 저장 (변수=변수>>피연산자 와 동일)
	변수	>>>=	피연산자	우측의 피연산자의 값과 변수의 값을 >>> 연산 후 결과를 변수에 저장 (변수=변수>>>피연산자 와 동일)

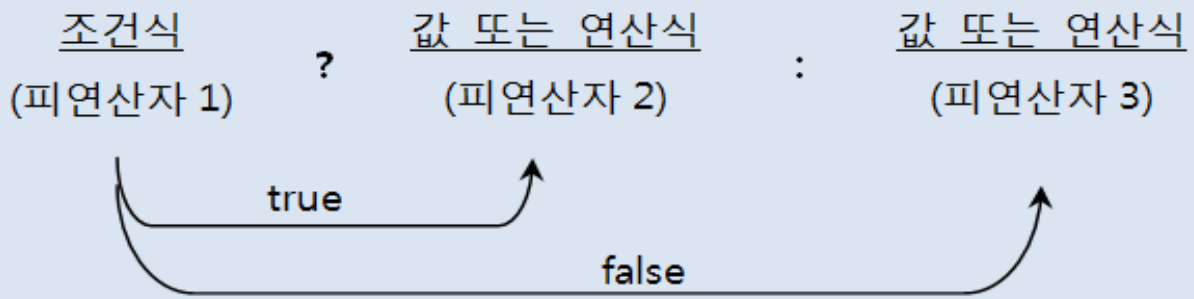
❖ 복합 대입 연산자: AssignmentOperatorExample.java

```
public class AssignmentOperatorExample {
   public static void main(String[] args) {
      int result = 0;

      result += 10;
      System.out.println("result=" + result);

      result -= 5;
      System.out.println("result=" + result);

      result *= 3;
      System.out.println("result=" + result);

      result /= 5;
      System.out.println("result=" + result);

      result %= 3;
      System.out.println("result=" + result);
   }
}
```

삼항 연산자

❖ 삼항 연산자란?

- 세 개의 피연산자를 필요로 하는 연산자
- 앞의 조건식 결과에 따라 콜론 앞 뒤의 피연산자 선택 → 조건 연산식




```
int score = 95;
char grade = (score>90) ?  'A'  :  'B'
```

=

```
int score = 95;
char grade;
if(score>90) {
   grade = 'A';
} else {
   grade = 'B';
}
```

삼항 연산자

❖ 복합 대입 연산자: ConditionalOperationExample.java

```
public class ConditionalOperationExample {
   public static void main(String[] args) {
      int score = 85;

      char grade = (score > 90) ? 'A' : ((score > 80) ? 'B' : 'C');

      System.out.println(score + "점은 " + grade + "등급입니다.");
   }
}
```