BREAK, CONTINUE, RETURN

break

- You can use a break to terminate a for, while, or do-while loop

```
public class Break_1 {
   public static void main(String[] args) {
      final int[] values = {10, -10, 20, -20, 30};

      int sum = 0 ;
      for ( final int value : values ) {
         if ( value < 0 ) break ;
         sum += value ;
      }
      System.out.println(sum); // 10
   }
}
```

break

```
import java.util.Scanner;

public class Break_2 {
   public static void main(String[] args) {
      final Scanner scanner = new Scanner(System.in);
      int sum = 0 ;
      while ( true ) {
         final int value = scanner.nextInt();
         if ( value <= 0 ) break;

         sum += value;
      }
      scanner.close();

      System.out.println("SUM: " + sum);
   }
}
```

continue

- The continue statement skips the current iteration of a for, while, or do loop.

```
public class Continue_1 {
   public static void main(String[] args) {
      final int[] values = {10, -10, 20, -20, 30};

      int sum = 0 ;
      for ( final int value : values ) {
         if ( value < 0 ) continue ;

         sum += value ;
      }
      System.out.println(sum); // 60(=10+20+30)
   }
}
```

continue

```
import java.util.Scanner;

public class Continue_2 {
   public static void main(String[] args) {
      final Scanner scanner = new Scanner(System.in);
      int sum = 0 ;
      while ( sum <= 10 ) {
         final int value = scanner.nextInt();
         if ( value <= 0 ) continue;

         sum += value;
      }
      scanner.close();

      System.out.println("SUM: " + sum);
   }
}
```

return

- The return statement exits from the current method, and control flow returns to where the method was invoked

```
public class Return_1 {
   public static void main(String[] args) {
      final int[] values = {10, -10, 20, -20, 30};

      final int sum = getSum(values);            // 10
      System.out.println(sum);
   }
   private static int getSum(final int[] intValues) {
      int sum = 0 ;
      for ( final int value : intValues ) {
         if ( value < 0 ) return sum;
         sum += value ;
      }
      return sum;
   }
}
```

ENUM

Enumerated Type: enum

- Enumerated type is used to specify a variable with a limited set of values.

```
enum Fruit {APPLE, GRAPE, PEAR, NO_FRUIT} ;
public class Enum_1 {
      public static void main(String[] args) {
      final Fruit apple = Fruit.APPLE ; // Fruit.valueOf("APPLE")
      System.out.println(apple) ;

      final String 사과 = getFruitKoreanName(apple);
      System.out.println(apple.name() + " is " + 사과 ) ;

      final Fruit fruit = getFruit(System.in);
      final String fruitName = getFruitKoreanName(fruit);
      System.out.println(fruit.name() + " is " + fruitName) ;
   }
```

```
APPLE
APPLE is 사과
pear
PEAR is 배
```

```
    private static String getFruitKoreanName(final Fruit myFruit) {
        String fruitName ;
        switch ( myFruit ) {
            case APPLE : fruitName = "사과" ; break ;
            case GRAPE : fruitName = "포도" ; break ;
            case PEAR : fruitName = "배" ; break ;
            default : fruitName = "모름" ; break ;
        }
        return fruitName;
    }

    private static Fruit getFruit(InputStream in) {
        final Scanner scanner = new Scanner(in);
        final String fruitName = scanner.next();

        Fruit fruit;
        try {
            fruit = Fruit.valueOf(fruitName.toUpperCase());
        }
        catch ( IllegalArgumentException e ) { fruit = Fruit.NO_FRUIT; }
        finally { scanner.close(); }
        return fruit;
    }
}
```

Enumerated Type: enum

- You can specify values of enum constants at the creation time

```
enum Currency {
   PENNY(1), NICKLE(5), DIME(10), QUARTER(25);
   private final int value;
   private Currency(final int value) { this.value = value; }
   public int getValue() { return value; }
}
public class Enum_2 {
   public static void main(String args[]) {
      final Currency usCoin = Currency.DIME;
      if ( usCoin == Currency.DIME ) {
         System.out.println("enum can be compared using ==");
      }
      for ( final Currency coin: Currency.values() ) {
         System.out.println(coin.name() + " "  + coin.getValue());
      }
   }
}
```

```
enum can be compared using ==
PENNY 1
NICKLE 5
DIME 10
QUARTER 25
```

Enumerated Type: enum

```
enum Fruit {
   APPLE("사과"), GRAPE("포도"), PEAR("배");

   private final String name;

   private Fruit(final String name) { this.name = name; }
   public String getName() { return name; }
}

public class Enum_3 {
   public static void main(String[] args) {
      final Fruit[] fruits = {Fruit.PEAR, Fruit.GRAPE, Fruit.APPLE, Fruit.APPLE};

      for ( final Fruit fruit : fruits )
         System.out.println("The fruit is " + fruit.getName() ) ;
   }
}
```

```
The fruit is 배
The fruit is 포도
The fruit is 사과
The fruit is 사과
```

```
import java.text.DecimalFormat;
enum ShoesKind {
   WALKING("워킹화", 100_000), RUNNING("러닝화", 200_000),
      TRACKING("트래킹화", 300_000); // Underscores in Numeric Literals since Java 7

   private final String name;
   private final int price;

   private ShoesKind(final String name, final int price) {
      this.name = name; this.price = price;
   }
   public String getName() { return name; }
   public int getPrice() { return price; }
}

public class Enum_4 {
   public static void main(String args[]) {
      final DecimalFormat priceFormat = new DecimalFormat("###,### ");
      final Currency currency = priceFormat.getCurrency();

      for ( final ShoesKind shoes: ShoesKind.values() ) {
         System.out.println(String.valueOf(shoes).toLowerCase() + " "
            + shoes.getName() + " : "
            + priceFormat.format(shoes.getPrice()) + currency);
      }
   }
}
```

```
walking 워킹화 : 100,000 KRW
running 러닝화 : 200,000 KRW
tracking 트래킹화 : 300,000 KRW
```

Q&A