Control Structures

- ❖Loop: while, for, do
- ❖Decision: if, switch
- ❖Branching: break, continue, return
- ❖enum

LOOP: WHILE, FOR, DO

Loop

- A **loop** statement allows us to execute a statement or group of statements multiple times

loop	Description
while	Repeats a statement or group of statements while a given condition is true
for	Execute a sequence of statements for a specific number of times
do	Like a while statement, except that it tests the condition at the end of the loop body

while loop

- A while loop statement repeatedly executes a target statement as long as a given condition is true

```
while ( condition ) {
   // Statements
}
```

while loop

```
public class WhileLoop_1 {
   public static void main(String args[]) {
      int x = 1;
      while ( x <=10 ) {
         System.out.printf("value of x : %d%n", x );
         x++;
      }
   }
}
```

```
value of x : 1
value of x : 2
value of x : 3
value of x : 4
value of x : 5
value of x : 6
value of x : 7
value of x : 8
value of x : 9
value of x : 10
```

while loop

```
public class WhileLoop_2 {
   public static void main(String args[]) {
      int sum = 0 ;
      int i = 1 ;
      while ( (i <= 10) && (sum < 30) ) {
         sum += i ;
         System.out.printf("Sum of 1 to %d: %d%n", i, sum);
         i ++ ;
      }
   }
}
```

```
Sum of 1 to 1: 1
Sum of 1 to 2: 3
Sum of 1 to 3: 6
Sum of 1 to 4: 10
Sum of 1 to 5: 15
Sum of 1 to 6: 21
Sum of 1 to 7: 28
Sum of 1 to 8: 36
```

while loop

```
import java.util.Scanner;
public class WhileLoop_3 {
   public static void main(String[] args) {
      final String inputString = "10 20 30 50";
      final Scanner scanner = new Scanner(inputString);

      int sum = 0;
      while ( scanner.hasNext() && (sum <= 50) ) {
         final int value = scanner.nextInt();
         sum += value;
      }
      scanner.close();

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

for loop

- A **for** loop is a repetition control structure that allows you to efficiently write a loop that needs to be executed a specific number of times.
- A **for** loop is useful when you know how many times a task is to be repeated

```
for ( init; condition; update ) {
   // Statements
}
```

for loop

```
public class ForLoop_1 {
   public static void main(String args[]) {
      for ( int i = 1; i <= 10; i ++ ) {
         System.out.printf("value of x : %d%n", i );
      }
   }
}
```

```
value of x : 1
value of x : 2
value of x : 3
value of x : 4
value of x : 5
value of x : 6
value of x : 7
value of x : 8
value of x : 9
value of x : 10
```

Enhanced for loop

- for statement also has another form designed for iteration through Collections and arrays

```
public class ForLoop_2 {
   public static void main(String args[]) {
      final int[] numbers = new int[10];
      for ( int i = 0; i < numbers.length; i ++ )
         numbers[i] = i+1;

      for ( final int i : numbers ) {
         System.out.printf("value of x : %d%n", i );
      }
   }
}
```

Enhanced for loop

```
import java.util.ArrayList;
import java.util.List;
public class ForLoop_3 {
   public static void main(String args[]) {
      final List<String> messages = new ArrayList<>();
      messages.add("Hello");
      messages.add("자바₩tGreat");
      messages.add("₩t10 !");

      int wordCount = 0;
      int charCount = 0;
      for ( final String message : messages ) {
         System.out.println(message);
         wordCount ++;
         charCount += message.trim().length();
      }
      System.out.println("Word: " + wordCount + ", Chars: " + charCount);
   }
}
```

```
Hello
자바      Great
          10 !
Word: 3, Chars: 17
```

Enhanced for loop

```
import java.util.ArrayList;
import java.util.List;

public class ForLoop_4 {
   public static void main(String args[]) {
      final List<String> messages = new ArrayList<>();

      messages.add("Hello");
      messages.add("Java");

      for ( final String message : messages ) {
         final char[] charArray = message.toCharArray();
         for ( final char aChar : charArray )
            System.out.print(Character.toUpperCase(aChar));
         System.out.println();
      }
   }
}
```

```
HELLO
JAVA
```

do loop

- A **do...while** loop is similar to a while loop, except that a do...while loop is guaranteed to execute at least one time

```
do {
   // Statements
} while ( condition ) ;
```

do loop

```
public class DoLoop_1 {
   public static void main(String args[]) {
      int x = 1;
      do {
         System.out.printf("value of x : %d%n", x );
         x++;
      } while ( x <= 10 );
   }
}
```

```
value of x : 1
value of x : 2
value of x : 3
value of x : 4
value of x : 5
value of x : 6
value of x : 7
value of x : 8
value of x : 9
value of x : 10
```

while loop vs do loop

```
public class Do_While_Compare {
   public static void main(String args[]) {
      int x = 0;
      int sum1 = 0;
      do {
         x ++ ;
         sum1 += x;
      } while (x < 10 ) ;
      System.out.println(sum1);    // 55

      int y = 0;
      int sum2 = 0;
      while ( y < 10 ) {
         y ++;
         sum2 += y;
      }
      System.out.println(sum2);    // 55
   }
}
```

while loop vs do loop

```
public class Do_While_Compare {
   public static void main(String args[]) {
      int x = 10;
      int sum1 = 0;
      do {
         x ++ ;
         sum1 += x;
      } while (x < 10 ) ;
      System.out.println(sum1);    // 11

      int y = 10;
      int sum2 = 0;
      while ( y < 10 ) {
         y ++;
         sum2 += y;
      }
      System.out.println(sum2);    // 0
   }
}
```

DECISION: IF, SWITCH

Decision

- Decision making structures have one or more conditions to be evaluated or tested by the program

loop	Description
if	An if statement consists of a boolean expression followed by one or more statements
switch	A switch statement allows a variable to be tested for equality against a list of values

if

```
import java.util.Scanner;

public class If_1 {
   public static void main(String[] args) {
      final Scanner scanner = new Scanner(System.in);

      final int testScore = scanner.nextInt();

      char grade = 'F';
      if ( testScore >= 90 ) {
         grade = 'A';
      }
      System.out.println("Grade = " + grade);

      scanner.close();
   }
}
```

if

```
import java.util.Scanner;

public class If_2 {
   public static void main(String[] args) {
      final Scanner scanner = new Scanner(System.in);

      final int testScore = scanner.nextInt();

      char grade;
      if ( testScore >= 90 ) {
         grade = 'A';
      }
      else {
         grade = 'F';
      }
      System.out.println("Grade = " + grade);

      scanner.close();
   }
}
```

if

```
public class If_3 {
   public static void main(String[] args) {
      final Scanner scanner = new Scanner(System.in);
      final int testScore = scanner.nextInt();

      char grade;
      if ( testScore >= 90 ) {
         grade = 'A';
      } else if ( testScore >= 80 ) {
         grade = 'B';
      } else if ( testScore >= 70 ) {
         grade = 'C';
      } else if ( testScore >= 60 ) {
         grade = 'D';
      } else {
         grade = 'F';
      }
      System.out.println("Grade = " + grade);
      scanner.close();
   }
}
```

```
public class If_4 {
   public static void main(String[] args) {
       final Scanner scanner = new Scanner(System.in);
       while ( true ) {
          final int testScore = scanner.nextInt();

          char grade;
          if ( testScore >= 90 ) {
              grade = 'A';
          } else if ( testScore >= 80 ) {
              grade = 'B';
          } else if ( testScore >= 70 ) {
              grade = 'C';
          } else if ( testScore >= 60 ) {
              grade = 'D';
          } else {
              grade = 'F';
          }
          System.out.println("Grade = " + grade);
          if ( grade == 'F' ) {
            System.out.println("BYE");
            break;
          }
       }
       scanner.close();
   }
}
```

```
public class If_5 {
 public static void main(String args[]) {
   final String message = "Java 8 !";
   for ( final char aChar : message.toCharArray() ) {
     final StringBuilder sb = new StringBuilder();
     sb.append(aChar + " : ");
     if ( Character.isDigit(aChar) )
       sb.append("digit.");
     else if ( Character.isLowerCase(aChar) )
       sb.append("lowercase.");
     else if ( Character.isUpperCase(aChar) )
       sb.append("uppercase.");
     else if ( Character.isWhitespace(aChar) )
       sb.append("whitespace.");
     else
       sb.append("neither alphanumeric nor whitespace.");
     System.out.println(sb.toString());
   }
 }
}
```

```
J : uppercase.
a : lowercase.
v : lowercase.
a : lowercase.
  : whitespace.
8 : digit.
  : whitespace.
!  : neither alphanumeric nor whitespace.
```

```
public class Switch_1 {
   public static void main(String[] args) {
        String monthStr = null;

        final int month = 8;
        switch (month) {
            case 1:  monthStr = "January"; break;
            case 2:  monthStr = "February"; break;
            case 3:  monthStr = "March"; break;
            case 4:  monthStr = "April"; break;
            case 5:  monthStr = "May"; break;
            case 6:  monthStr = "June"; break;
            case 7:  monthStr = "July"; break;
            case 8:  monthStr = "August"; break;
            case 9:  monthStr = "September"; break;
            case 10: monthStr = "October"; break;
            case 11: monthStr = "November"; break;
            case 12: monthStr = "December"; break;
            default: break;
        }
        System.out.println(monthStr); // August
    }
}
```

switch

```
import java.util.Arrays;
import java.util.List;

public class Switch_2 {
   public static void main(String[] args) {
      final String[] monthStrs = {"January", "February", "March", "April",
         "May", "June", "July", "August", "September", "October",
         "November", "December"};

      final int month = 8;

      System.out.println(monthStrs[month-1]); // August

      final List<String> monthList = Arrays.asList(monthStrs);
      System.out.println(monthList.get(month-1)); // August
    }
}
```

```
public class Switch_3 {
   public static void main(String[] args) {
        final List<String> futureMonths = new ArrayList<>();
        final int month = 8;
        switch (month) {
            case 1:  futureMonths.add("January");
            case 2:  futureMonths.add("February");
            case 3:  futureMonths.add("March");
            case 4:  futureMonths.add("April");
            case 5:  futureMonths.add("May");
            case 6:  futureMonths.add("June");
            case 7:  futureMonths.add("July");
            case 8:  futureMonths.add("August");
            case 9:  futureMonths.add("September");
            case 10: futureMonths.add("October");
            case 11: futureMonths.add("November");
            case 12: futureMonths.add("December");
                     break;
            default: break;
        }
        for ( final String monthName : futureMonths )
           System.out.println(monthName);
   }
}
```

```
August
September
October
November
December
```

String in Switch Case

- Since Java 7(2011), String is allowed in the expression of a switch statement

```
public class Switch_4 {
 public static void main(String[] args) {
   final String dayOfWeek = args[0];
   final String typeOfDay = getTypeOfDay(dayOfWeek);
   System.out.printf("%10s is %20s%n", dayOfWeek, typeOfDay);
 }
```

String in Switch Case

```
private static String getTypeOfDay(final String dayOfWeek) {
  String typeOfDay;
  switch ( dayOfWeek.toUpperCase() ) {
    case "MONDAY": typeOfDay = "Start of work week"; break;
    case "TUESDAY":
    case "WEDNESDAY":
    case "THURSDAY": typeOfDay = "Midweek"; break;
    case "FRIDAY": typeOfDay = "End of work week"; break;
    case "SATURDAY":
    case "SUNDAY": typeOfDay = "Weekend"; break;
    default:
      typeOfDay = "Invalid day of the week”;
      break;
  }
  return typeOfDay;
 }
}
```

```
import java.util.HashMap;
import java.util.Map;
public class Switch_5 {
 private static final Map<String, String> typeOfDayMap = new HashMap<>();

 static {
   typeOfDayMap.put("MONDAY", "Start of work week");
   typeOfDayMap.put("TUESDAY", "Midweek");
   typeOfDayMap.put("WEDNESDAY", "Midweek");
   typeOfDayMap.put("THURSDAY", "Midweek");
   typeOfDayMap.put("FRIDAY", "End of work week");
   typeOfDayMap.put("SATURDAY", "Weekend");
   typeOfDayMap.put("SUNDAY", "Weekend");
 }

 public static void main(String[] args) {
   final String dayOfWeek = args[0];
   final String typeOfDay = getTypeOfDay(dayOfWeek);
   System.out.printf("%10s is %20s%n", dayOfWeek, typeOfDay);
 }
 private static String getTypeOfDay(final String dayOfWeek) {
   final String typeOfDay = typeOfDayMap.get(dayOfWeek.toUpperCase());
   return ( typeOfDay != null ) ? typeOfDay : "Invalid day of the week";
 }
}
```

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