Java Basics to OOP

1. Class and Object

1-1. Class

```
public class Bicycle {

    // the Bicycle class has
    // three fields
    public int cadence; //분당 페달 회전수
    public int gear; // 기어; 변속장치
    public int speed;

    // the Bicycle class has
    // one constructor
    public Bicycle(int startCadence, int startSpeed, int startGear) {
        gear = startGear;
        cadence = startCadence;
        speed = startSpeed;
    }

    // the Bicycle class has
    // four methods
    public void setCadence(int newValue) {
        cadence = newValue;
    }

    public void setGear(int newValue) {
        gear = newValue;
    }

    public void applyBrake(int decrement) {
        speed -= decrement;
    }

    public void speedUp(int increment) {
        speed += increment;
    }

}
```

1-2. Extends

```
public class MountainBike extends Bicycle {

    // the MountainBike subclass has
    // one field
    public int seatHeight;

    // the MountainBike subclass has
    // one constructor
    public MountainBike(int startHeight, int startCadence,
                        int startSpeed, int startGear) {
        super(startCadence, startSpeed, startGear);
        seatHeight = startHeight;
    }

    // the MountainBike subclass has
    // one method
    public void setHeight(int newValue) {
        seatHeight = newValue;
    }

}
```

2. Variables - Primitive types and Arrays

2-1. Integer Literals

```
// The number 26, in decimal
int decVal = 26;
//  The number 26, in hexadecimal
int hexVal = 0x1a;
// The number 26, in binary
int binVal = 0b11010;
```

2-2. Floating point literals

```
double d1 = 123.4;
// same value as d1, but in scientific notation
double d2 = 1.234e2;
float f1  = 123.4f;
```

2-3. Array

```
class ArrayDemo {
    public static void main(String[] args) {
        // declares an array of integers
        int[] anArray;

        // allocates memory for 10 integers
        anArray = new int[10];

        // initialize first element
        anArray[0] = 100;
        // initialize second element
        anArray[1] = 200;
        // and so forth
        anArray[2] = 300;
        anArray[3] = 400;
        anArray[4] = 500;
        anArray[5] = 600;
        anArray[6] = 700;
        anArray[7] = 800;
        anArray[8] = 900;
        anArray[9] = 1000;

        System.out.println("Element at index 0: "
                           + anArray[0]);
        System.out.println("Element at index 1: "
                           + anArray[1]);
        System.out.println("Element at index 2: "
                           + anArray[2]);
        System.out.println("Element at index 3: "
                           + anArray[3]);
        System.out.println("Element at index 4: "
                           + anArray[4]);
        System.out.println("Element at index 5: "
                           + anArray[5]);
        System.out.println("Element at index 6: "
                           + anArray[6]);
        System.out.println("Element at index 7: "
                           + anArray[7]);
        System.out.println("Element at index 8: "
                           + anArray[8]);
        System.out.println("Element at index 9: "
                           + anArray[9]);
    }
}
```

2-4. Declaring Arrays

```
byte[] anArrayOfBytes;
short[] anArrayOfShorts;
long[] anArrayOfLongs;
float[] anArrayOfFloats;
double[] anArrayOfDoubles;
```

```
boolean[] anArrayOfBooleans;
char[] anArrayOfChars;
String[] anArrayOfStrings;
```

2-5. Initializing Arrays

```
 // create an array of integers
int [] anArray = new int[10];

anArray[0] = 100; // initialize first element
anArray[1] = 200; // initialize second element
anArray[2] = 300; // and so forth

//Alternatively, you can use the shortcut syntax to create and
// initialize an array:
int[] anArray = {
    100, 200, 300,
    400, 500, 600,
    700, 800, 900, 1000
};
```

2-6. Multidimensional Arrays

```
class MultiDimArrayDemo {
    public static void main(String[] args) {
        String[][] names = {
            {"Mr. ", "Mrs. ", "Ms. "},
            {"Smith", "Jones"}
        };
        // Mr. Smith
        System.out.println(names[0][0] + names[1][0]);
        // Ms. Jones
        System.out.println(names[0][2] + names[1][1]);
    }
}
```

2-7. Copying Arrays

```
class ArrayCopyDemo {
    public static void main(String[] args) {
        char[] copyFrom = { 'd', 'e', 'c', 'a', 'f', 'f', 'e',
          'i', 'n', 'a', 't', 'e', 'd' };
        char[] copyTo = new char[7];
```

```
        System.arraycopy(copyFrom, 2, copyTo, 0, 7);
        System.out.println(new String(copyTo));
    }
}
```

2-8. Array Manipulations

```
class ArrayCopyOfDemo {
    public static void main(String[] args) {

        char[] copyFrom = {'d', 'e', 'c', 'a', 'f', 'f', 'e',
            'i', 'n', 'a', 't', 'e', 'd'};

        char[] copyTo = java.util.Arrays.copyOfRange(copyFrom, 2, 9);

        System.out.println(new String(copyTo));
    }
}
```

3. Operations

```
class ArithmeticDemo {

    public static void main (String[] args) {

        int result = 1 + 2;
        // result is now 3
        System.out.println("1 + 2 = " + result);
        int original_result = result;

        result = result - 1;
        // result is now 2
        System.out.println(original_result + " - 1 = " + result);
        original_result = result;

        result = result * 2;
        // result is now 4
        System.out.println(original_result + " * 2 = " + result);
        original_result = result;

        result = result / 2;
        // result is now 2
        System.out.println(original_result + " / 2 = " + result);
        original_result = result;

        result = result + 8;
```

```
        // result is now 10
        System.out.println(original_result + " + 8 = " + result);
        original_result = result;

        result = result % 7;
        // result is now 3
        System.out.println(original_result + " % 7 = " + result);
    }
}
```

3-1. Concatenating Strings

```
class ConcatDemo {
    public static void main(String[] args){
        String firstString = "This is";
        String secondString = " a concatenated string.";
        String thirdString = firstString+secondString;
        System.out.println(thirdString);
    }
}
```

3-2. Unary Operators

```
class UnaryDemo {

    public static void main(String[] args) {

        int result = +1;
        // result is now 1
        System.out.println(result);

        result--;
        // result is now 0
        System.out.println(result);

        result++;
        // result is now 1
        System.out.println(result);

        result = -result;
        // result is now -1
        System.out.println(result);

        boolean success = false;
        // false
        System.out.println(success);
        // true
        System.out.println(!success);
```

```
    }
}
```

3-3. Prefix and Postfix - increment and decrement operators

```
class PrePostDemo {
    public static void main(String[] args){
        int i = 3;
        i++;
        // prints 4
        System.out.println(i);
        ++i;
        // prints 5
        System.out.println(i);
        // prints 6
        System.out.println(++i);
        // prints 6
        System.out.println(i++);
        // prints 7
        System.out.println(i);
    }
}
```

3-4. Equality and Relational Operators

```
class ComparisonDemo {

    public static void main(String[] args){
        int value1 = 1;
        int value2 = 2;
        if(value1 == value2)
            System.out.println("value1 == value2");
        if(value1 != value2)
            System.out.println("value1 != value2");
        if(value1 > value2)
            System.out.println("value1 > value2");
        if(value1 < value2)
            System.out.println("value1 < value2");
        if(value1 <= value2)
            System.out.println("value1 <= value2");
    }
}
```

3-5. Conditional Operators

```
class ConditionalDemo1 {

    public static void main(String[] args){
        int value1 = 1;
        int value2 = 2;
        if((value1 == 1) && (value2 == 2))
            System.out.println("value1 is 1 AND value2 is 2");
        if((value1 == 1) || (value2 == 1))
            System.out.println("value1 is 1 OR value2 is 1");
    }
}
```

3-6. Ternary Operator ?:

```
class ConditionalDemo2 {

    public static void main(String[] args){
        int value1 = 1;
        int value2 = 2;
        int result;
        boolean someCondition = true;
        result = someCondition ? value1 : value2;

        System.out.println(result);
    }
}
```

3-7. Type Comparison Operator `instanceof`

```
class InstanceofDemo {
    public static void main(String[] args) {

        Parent obj1 = new Parent();
        Parent obj2 = new Child();

        System.out.println("obj1 instanceof Parent: "
            + (obj1 instanceof Parent));
        System.out.println("obj1 instanceof Child: "
            + (obj1 instanceof Child));
        System.out.println("obj1 instanceof MyInterface: "
            + (obj1 instanceof MyInterface));
        System.out.println("obj2 instanceof Parent: "
            + (obj2 instanceof Parent));
        System.out.println("obj2 instanceof Child: "
            + (obj2 instanceof Child));
        System.out.println("obj2 instanceof MyInterface: "
            + (obj2 instanceof MyInterface));
```

```
    }
}

class Parent {}
class Child extends Parent implements MyInterface {}
interface MyInterface {}
```

▼ When using the `instanceof` operator, keep in mind that `null` is not an instance of anything.

3-8. Bitwise and Bit Shift Operators

```
class BitDemo {
    public static void main(String[] args) {
        int bitmask = 0x000F;
        int val = 0x2222;
        // prints "2"
        System.out.println(val & bitmask);
    }
}
```

4. Control Flow Statements

4-1. If - then statement

```
void applyBrakes() {
    // the "if" clause: bicycle must be moving
    if (isMoving){
        // the "then" clause: decrease current speed
        currentSpeed--;
    }
}
```

Opening and Closing braces are optional, given taht the "then" clause contains only one statement.

```
void applyBrakes() {
    // same as above, but without braces
    if (isMoving)
        currentSpeed--;
}
```

4-2. if - then - else statement

```
void applyBrakes() {
    if (isMoving) {
        currentSpeed--;
    } else {
        System.err.println("The bicycle has already stopped!");
    }
}
```

4-3. if - else demo

```
class IfElseDemo {
    public static void main(String[] args) {

        int testscore = 76;
        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);
    }
}
```

4-4. switch statement

```
public class SwitchDemo {
    public static void main(String[] args) {

        int month = 8;
        String monthString;
        switch (month) {
            case 1:  monthString = "January";
                     break;
```

```
            case 2:  monthString = "February";
                     break;
            case 3:  monthString = "March";
                     break;
            case 4:  monthString = "April";
                     break;
            case 5:  monthString = "May";
                     break;
            case 6:  monthString = "June";
                     break;
            case 7:  monthString = "July";
                     break;
            case 8:  monthString = "August";
                     break;
            case 9:  monthString = "September";
                     break;
            case 10: monthString = "October";
                     break;
            case 11: monthString = "November";
                     break;
            case 12: monthString = "December";
                     break;
            default: monthString = "Invalid month";
                     break;
        }
        System.out.println(monthString);
    }
}
```

Each `break;` statement terminates the enclosing `switch` statement.

4-5. Fall through switch statement

```
public class SwitchDemoFallThrough {

    public static void main(String[] args) {
        java.util.ArrayList<String> futureMonths =
            new java.util.ArrayList<String>();

        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;
        }

        if (futureMonths.isEmpty()) {
            System.out.println("Invalid month number");
        } else {
            for (String monthName : futureMonths) {
               System.out.println(monthName);
            }
        }
    }
}
```

4-6. Multiple case labels allowed

```
class SwitchDemo2 {
    public static void main(String[] args) {

        int month = 2;
        int year = 2000;
        int numDays = 0;

        switch (month) {
            case 1: case 3: case 5:
            case 7: case 8: case 10:
            case 12:
                numDays = 31;
                break;
            case 4: case 6:
            case 9: case 11:
                numDays = 30;
                break;
            case 2:
                if (((year % 4 == 0) &&
                     !(year % 100 == 0))
                     || (year % 400 == 0))
                    numDays = 29;
                else
                    numDays = 28;
                break;
            default:
                System.out.println("Invalid month.");
                break;
        }
        System.out.println("Number of Days = "
                           + numDays);
    }
}
```

4-7. Strings allowed as a switch statement's expression

```
public class StringSwitchDemo {

    public static int getMonthNumber(String month) {

        int monthNumber = 0;

        if (month == null) {
            return monthNumber;
        }

        switch (month.toLowerCase()) {
            case "january":
                monthNumber = 1;
                break;
            case "february":
                monthNumber = 2;
                break;
            case "march":
                monthNumber = 3;
                break;
            case "april":
                monthNumber = 4;
                break;
            case "may":
                monthNumber = 5;
                break;
            case "june":
                monthNumber = 6;
                break;
            case "july":
                monthNumber = 7;
                break;
            case "august":
                monthNumber = 8;
                break;
            case "september":
                monthNumber = 9;
                break;
            case "october":
                monthNumber = 10;
                break;
            case "november":
                monthNumber = 11;
                break;
            case "december":
                monthNumber = 12;
                break;
            default:
                monthNumber = 0;
                break;
        }
```

```
        return monthNumber;
    }

    public static void main(String[] args) {

        String month = "August";

        int returnedMonthNumber =
            StringSwitchDemo.getMonthNumber(month);

        if (returnedMonthNumber == 0) {
            System.out.println("Invalid month");
        } else {
            System.out.println(returnedMonthNumber);
        }
    }
}
```

4-8. While and do~ while statements

```
class WhileDemo {
    public static void main(String[] args){
        int count = 1;
        while (count < 11) {
            System.out.println("Count is: " + count);
            count++;
        }
    }
}
```

```
class DoWhileDemo {
    public static void main(String[] args){
        int count = 1;
        do {
            System.out.println("Count is: " + count);
            count++;
        } while (count < 11);
    }
}
```

4-9. The for statement

```
for (initialization; termination; increment) {
    statement(s)
}
```

When using this version of the `for` statement, keep in mind that:

- The *initialization* expression initializes the loop; it's executed once, as the loop begins.
- When the *termination* expression evaluates to `false`, the loop terminates.
- The *increment* expression is invoked after each iteration through the loop; it is perfectly acceptable for this expression to increment decrement a value.

  or

```
class ForDemo {
    public static void main(String[] args){
         for(int i=1; i<11; i++){
              System.out.println("Count is: " + i);
         }
    }
}
```

5. Branching Statements

5-1. The `break` statement

```
class BreakDemo {
    public static void main(String[] args) {

        int[] arrayOfInts =
            { 32, 87, 3, 589,
              12, 1076, 2000,
              8, 622, 127 };
        int searchfor = 12;

        int i;
        boolean foundIt = false;

        for (i = 0; i < arrayOfInts.length; i++) {
            if (arrayOfInts[i] == searchfor) {
                foundIt = true;
                break;
            }
        }

        if (foundIt) {
            System.out.println("Found " + searchfor + " at index " + i);
        } else {
```

```
            System.out.println(searchfor + " not in the array");
        }
    }
}
```

▼ An unlabeled break statement terminates the innermost switch, for, while, or do-while statement, but a labeled break terminates an outer statement.

```
class BreakWithLabelDemo {
    public static void main(String[] args) {

        int[][] arrayOfInts = {
            { 32, 87, 3, 589 },
            { 12, 1076, 2000, 8 },
            { 622, 127, 77, 955 }
        };
        int searchfor = 12;

        int i;
        int j = 0;
        boolean foundIt = false;

    search:
        for (i = 0; i < arrayOfInts.length; i++) {
            for (j = 0; j < arrayOfInts[i].length;
                 j++) {
                if (arrayOfInts[i][j] == searchfor) {
                    foundIt = true;
                    break search;
                }
            }
        }

        if (foundIt) {
            System.out.println("Found " + searchfor + " at " + i + ", " + j);
        } else {
            System.out.println(searchfor + " not in the array");
        }
    }
}
```

5-2. The `continue` statement

▼ The continue statement skips the current iteration of a for, while , or do-while loop. The unlabeled form skips to the end of the innermost loop's body and evaluates the boolean expression that controls the loop.

```
class ContinueDemo {
    public static void main(String[] args) {

        String searchMe = "peter piper picked a " + "peck of pickled peppers";
        int max = searchMe.length();
        int numPs = 0;

        for (int i = 0; i < max; i++) {
            // interested only in p's
            if (searchMe.charAt(i) != 'p')
                continue;

            // process p's
            numPs++;
        }
        System.out.println("Found " + numPs + " p's in the string.");
    }
}
```

```
class ContinueWithLabelDemo {
    public static void main(String[] args) {

        String searchMe = "Look for a substring in me";
        String substring = "sub";
        boolean foundIt = false;

        int max = searchMe.length() -
                  substring.length();

    test:
        for (int i = 0; i <= max; i++) {
            int n = substring.length();
            int j = i;
            int k = 0;
            while (n-- != 0) {
                if (searchMe.charAt(j++) != substring.charAt(k++)) {
                    continue test;
                }
            }
            foundIt = true;
                break test;
        }
        System.out.println(foundIt ? "Found it" : "Didn't find it");
    }
}
```

5-3. The `return` statement

▼ The last of the branching statements is the return statement. The return statement **exits from the current method,** and control flow returns to where the

method was invoked.

▼ The **data type of the returned value** must match the type of the **method's declared return value.** When a method is declared void, use the form of return that doesn't return a value.

```
return; //returning nothing. void.
```

6. Classes and Objects

6-1. Passing Information to a Method or a Constructor

```
public double computePayment(
                  double loanAmt,
                  double rate,
                  double futureValue,
                  int numPeriods) {
    double interest = rate / 100.0;
    double partial1 = Math.pow((1 + interest),
                    - numPeriods);
    double denominator = (1 - partial1) / interest;
    double answer = (-loanAmt / denominator)
                    - ((futureValue * partial1) / denominator);
    return answer;
}
```

- Parameter : Method 선언시에 선언부에 정의하는 것이 Parameter.
- Arguments : Method를 사용할 때 정의된 Parameter들에 해당하는 값을 넣는데, 그 값들을 argument 라고 한다.

6-2. Arbitrary number of arguments

Varargs ⇒ Variable (변할 수 있는) Arguments

```
public Polygon polygonFrom(Point... corners) {
    int numberOfSides = corners.length;
    double squareOfSide1, lengthOfSide1;
    squareOfSide1 = (corners[1].x - corners[0].x)
                    * (corners[1].x - corners[0].x)
```

```
                    + (corners[1].y - corners[0].y)
                    * (corners[1].y - corners[0].y);
    lengthOfSide1 = Math.sqrt(squareOfSide1);

    // more method body code follows that creates and returns a
    // polygon connecting the Points
}
```

▼ 인자값이 몇 개가 넘어올 지 모르는 경우 Varargs 를 사용할 수 있다.

`Point... corners` 는 Point [ ] corners 와 동일하다. 배열을 만드는 short cut.

▼ 다각형을 만드는 메소드의 인자값 → 몇 개의 인자가 들어올 지 모르므로 Varargs 를 사용하였다.

cf. Varargs 의 흔한 예

```
public PrintStream printf(String format, Object... args)
```

⇒ 처음 인자값으로는 포맷 스트링이 들어가고, 두 번 째 인자값으로는 여러개의 스트링 값이 들어간다.

```
System.out.printf("%s: %d, %s%n", name, idnum, address);
```

⇒ 이처럼 세 개의 값을 넣어서 format 스트링에 맞게 출력할 수 도 있고

```
System.out.printf("%s: %d, %s, %s, %s%n", name, idnum, address, phone, email);
```

⇒ 더 많은 값을 넣을 수 도 있다.

6-3. **Parameter names**

Method 혹은 Constructor에 Parameter 를 선언할 때 해당 매개변수에 이름을 부여한다. 이 이름은 method body 내에서 넘겨진 인자값을 참조(refer)하기 위해 사용된다.

▼ 주의할 점은 매개변수의 이름은 메소드 밖의 변수들과 이름이 같아도 된다. 이런 면에서 매개변수는 field 를 shadow 한다고 하기도 한다. Field 들을 shadowing 하는 것은 코드의 가독성을 낮추는 요인이 되므로, 주로 생성자의 매개변수 혹은 setter의 매개변수로만 메소드 밖의 변수와 같은 이름을 쓰는 것이 관례이다.

```
public class Circle {
    private int x, y, radius;

    // Constructor 의 매개변수는 통상 클래스 안에 정의된 다른 필드들과
    // 같은 이름을 가진다.
    public Circle(int x, int y, int radius){
        this.x = x;
        this.y = y;
        this.radius = radius;
    }

    // setter 에 쓰이는 경우도 마찬가지이다.
    public void setOrigin(int x, int y) {
        ...
    }
}
```

6-4. Passing Primitive Data Type Arguments : Pass by value

⇒ 매개변수로넘어간 기본 자료형 인자값들은 오직 그 메소드 안에서만 유효하다.

```
public class PassPrimitiveByValue {

    public static void main(String[] args) {

        int x = 3;

        // invoke passMethod() with
        // x as argument
        passMethod(x);

        // print x to see if its
        // value has changed
        System.out.println("After invoking passMethod, x = " + x);

    }

    // change parameter in passMethod()
    public static void passMethod(int p) {
        p = 10;
    }
}
```

위의 예제에서 passMethod(x) 는 매개변수로 넘어온 인자값을 10으로 바꾸고 있다. 하지만 매개변수로 x의 값이 넘어갈 때 x의 값이 복사가 되어 넘어가기 때문에 passMethod(int p) 안에서 p =10 을 할당한다 하더라도 소용이 없다. 따라서 최종적으로 출력되는 x의 값은 instance variable 로 선언된 3이다.

6-5. Passing Reference Data Type Arguments

```
public void moveCircle(Circle circle, int deltaX, int deltaY) {
    // code to move origin of circle to x+deltaX, y+deltaY
    circle.setX(circle.getX() + deltaX);
    circle.setY(circle.getY() + deltaY);

    // code to assign a new reference to circle
    circle = new Circle(0, 0);
}
```

위의 메소드에 아래와 같이 인자값을 넘겨 주었다고 하자.

```
moveCircle(myCircle, 23, 56)
```

메소드 안에서 circle 이라는 매개변수는 myCircle 의 주소값을 가리키게 된다. 하지만 마지막에 new Circle(0, 0); 를 해 준 결과 circle 은 더 이상 myCircle 의 주소값을 가리키는 것이 아니라, 메모리에 새로 생성된 new Circle(0, 0) 라는 객체의 주소값을 가리키게 된다. 따라서 myCircle 의 값이 변하는 것이 아니라, 단지 매개변수 circle이 가리키는 객체의 주소만이 달라졌을 뿐이다.

7. More on Classes

7-1. Understanding Class Members

Instance Variables : 만들어지는 객체마다 고유한 값을 갖게 되는 변수들. 예를들어 Person 객체의 String name 이라는 필드는 만들어지는 Person 객체마다 고유한 값을 갖게 될 것이다.

```
public class Person{

    private String name;
    ...
    public Person(String name){
      this.name = name;
    }

    public String getName(){
      return name;
    }
}
```

```
public class Main{

  public static void main(String [] args){
      Person me = new Person("my name");
      // me.getName() = "my name"; 이라는 값으로 초기화 될 것이다.
    }

}
```

한편 클래스를 만들 때 아무리 많은 객체가 만들어 지더라도 그 객체들이 공통적으로 갖게 되는 속성이 있을 수 있다. 예를들어 모든 사람은 다리가 2개라고 치자.

```
public class Person{
    private String name;
    private static final int NUM_OF_LEGS = 2;
    // 이와같이 static 변수로 선언하면, 모든 객체는 공통적으로 똑같이 이 변수의
    // 값을 갖게 된다.
}
```

static 변수는 참조하기 위해서 객체를 생성할 필요가 없기 때문에 클래스 이름으로 바로 접근이 가능하다. static method 도 마찬가지 이다.

```
// main 함수 안에서
System.out.println(Person.NUM_OF_LEGS);
// 숫자 2가 출력될 것이다.
```

그렇다고 해서 반드시 객체를 생성하지 않고 static 변수에 접근해야 하는 것은 아니다. 어차피 객체 모두가 공통적으로 갖고 있는 성질이므로

```
// main 함수 안에서
Person person = new Person("my name");
System.out.println(person.NUM_OF_LEGS);
//와 같이 접근해도 된다.
```

static field 뿐만 아니라 static method도 있다. 마찬가지로 객체를 생성하지 않아도 사용할 수 있는 method이다.

▼ 안드로이드에서는 보통 StringUtils / NetworkUtils 와 같이 Utility class 를 많이 작성하는데, 이런 경우 클래스 안에 static method 들을 정의한다.

▼ 예를들어

```
public class StringUtils{

  // This method concatenates "Mr. " to a name passed by the parameter.
  public static String concatMister(String name){
      return "Mr. " + name;
  }

}
```

이와같은 StringUtils.java 파일이 있었다고 하자. 이 때 concatMister가 하는 역할은 단지 매개변수로 받은 이름에 "Mr. " 라는 스트링을 붙여 줄 뿐이다. 이렇게 보조적인 역할을 해주는 메소드들을 따로 빼서 Util클래스에 정의한다. 이러한 클래스들은 굳이 고유한 객체 여러개를 생성해서 사용 할 필요 없이 이름 앞에 Mr. 라는 스트링만 붙여 주면 제 역할을 다 하는 것이기 때문에 클래스 내부에 static method 로 정의를 한다.

```
String myName = "Alex" ;

StringUtils.concatMister(myName);
// Mr. Alex 를 리턴할 것이다.
```

⇒ 하나의 스트링만 있으면 이런 과정이 번거롭게 느껴질 수 있지만, 예를들어 아주 많은 이름이 들어있는 배열이 있고, 그 이름들 모두에 Mr. 스트링을 붙여줘야 한다고 할 때 StringUtils 클래스의 유용함을 느낄 수 있다.

```
String [] manyNames = {"name1", "name2" .... "name1000"};

for(int i = 0; i < manyNames.length; i++){
  StringUtils.concatMister(manyNames[i]);
}

// 1000개의 이름들에 Mr. 를 concatenate 시킬 수 있다.
```

▼ 안드로이드에서 Constants 들을 별도의 클래스도 빼 놓는 것도 흔하다.

```
public class Constants{

    public static final String BASE_URL = "www.thisisasampleurl.com/124234234";

    public static final String PARCEL_KEY = "keyToTheData";
```

```
    public static final int NETWORK_TIMEOUT = 5000; //milli seconds

}
```

이와 같이 오타를 내기 쉬운 URL 스트링 이라든지, 혹은 어떠한 value 를 retrieve 하기 위한 고유한 key값을 상수로 선언한다.

위에서 NETWORK_TIMEOUT 을 굳이 상수로 선언한 이유는 코드를 읽는 사람 입장에서 5000이라는 숫자가 어떤 것을 의미하는 지 분명히 해 주는 장점도 있을 뿐더러, 코드에 반복적으로 등장하는 숫자라서 계속 써주기 귀찮아서 (?) 그런 것일 수 도 있다.

```
Uri.parseUrl(BASE_URL);
Movie movie = getParcelableExtra(PARCEL_KEY);
Network.setTimeOut(NETWORK_TIMEOUT);
//이런 식으로 쓰일 수 있다.
```

8. What is an Interface?

8-1.

메소드의 구현부 없이 선언부만 적어둔 것.

```
interface Bicycle {

    //  wheel revolutions per minute
    void changeCadence(int newValue);

    void changeGear(int newValue);

    void speedUp(int increment);

    void applyBrakes(int decrement);
}
```

위의 인터페이스를 구현하기 위해서 아래와 같이 새로운 클래스를 만들고 `implements` 키워드를 사용한다.

```
public class ACMEBicycle implements Bicycle {

    int cadence = 0;
    int speed = 0;
    int gear = 1;

   // The compiler will now require that methods
   // changeCadence, changeGear, speedUp, and applyBrakes
   // all be implemented. Compilation will fail if those
   // methods are missing from this class.

    void changeCadence(int newValue) {
         cadence = newValue;
    }

    void changeGear(int newValue) {
         gear = newValue;
    }

    void speedUp(int increment) {
         speed = speed + increment;
    }

    void applyBrakes(int decrement) {
         speed = speed - decrement;
    }

    void printStates() {
         System.out.println("cadence:" +
             cadence + " speed:" +
             speed + " gear:" + gear);
    }
}
```

8-2. Using Interface as a type

(Android Code 주의) 아래는 평범한 Movie class 이다.

```
public class Movie {

    private String movieName;
    private String releaseDate;
    private float votingRate;

    public Movie(String movieName, String releaseDate, float votingRate) {
        this.movieName = movieName;
        this.releaseDate = releaseDate;
        this.votingRate = votingRate;
    }
```

```
    public String getMovieName() {
        return movieName;
    }

    public String getReleaseDate() {
        return releaseDate;
    }

    public float getVotingRate() {
        return votingRate;
    }

}
```

영화 객체를 Main화면에서 Detail 화면으로 보내주는 상황을 생각해보자. 이런 경우 보통 사용하는 것이 Intent 클래스이다. 하지만 Movie는 우리가 정의한 Custom type 이기 때문에 Intent에 담아서 다른 화면으로 전송 할 수가 없다.

예를들어,

```
//Main 화면에서 Detail 화면으로 정보를 전송 할 것이다.
Intent intent = new Intent(MainActivity.this, DetailActivity.class);
//전송할 정보는 putExtra method 의 인자값으로 넣는다.
intent.putExtra(PARCEL_KEY, Parcelable parcel);
//내가 정의한 intent 를 갖고 화면 전환 액션을 취한다.
startActivity(intent);
```

Main화면에서 Detail 화면으로 정보를 넘겨줄 때 사용하는 putExtra() 메소드는 인자값으로 Primitive Data Type, Arrays of Primitive Data Type, Serializable, Parcelable 밖에 넘겨받지 못한다. 따라서 우리가 정의한 Custom type인 Movie 를 Parcelable interface 를 구현함으로써 Type을 Parcelable 로 확장시켜 주는 것이다.

▼ Parcelable 을 구현한 클래스는 Parcelable Type 으로 인식되기 때문에 Intent에 담아서 정보를 전달할 수 있다.

```
import android.os.Parcel;
import android.os.Parcelable;

public class Movie implements Parcelable {

    private String movieName;
    private String releaseDate;
    private float votingRate;

    public Movie(String movieName, String releaseDate, float votingRate) {
        this.movieName = movieName;
        this.releaseDate = releaseDate;
```

```
        this.votingRate = votingRate;
    }

// Parcel 객체를 다시 Movie 객체로 변환할 수 있게 해주는 메소드
    protected Movie(Parcel in) {
        movieName = in.readString();
        releaseDate = in.readString();
        votingRate = in.readFloat();
    }

// Parcel에다가 Movie 객체의 정보를 넘겨주는 메소드
    @Override
    public void writeToParcel(Parcel dest, int flags) {
        dest.writeString(movieName);
        dest.writeString(releaseDate);
        dest.writeFloat(votingRate);
    }

    @Override
    public int describeContents() {
//별로 하는 것 없는 메소드. 객체의 hashcode 를 반환해 준다.
        return 0;
    }

// Parcel로 감싸져 있던 Movie 객체를 다시 원상복귀 시켜주는 CREATOR
    public static final Creator<Movie> CREATOR = new Creator<Movie>() {
        @Override
        public Movie createFromParcel(Parcel in) {
          //우리가 원하던 Movie 객체
            return new Movie(in);
        }

        @Override
        public Movie[] newArray(int size) {
            return new Movie[size];
        }
    };

    public String getMovieName() {
        return movieName;
    }

    public String getReleaseDate() {
        return releaseDate;
    }

    public float getVotingRate() {
        return votingRate;
    }

}
```

Parcelable interface 를 보면 아래와 같다.

```
package android.os;

public interface Parcelable {
    int CONTENTS_FILE_DESCRIPTOR = 1;
    int PARCELABLE_WRITE_RETURN_VALUE = 1;

    int describeContents();

    void writeToParcel(Parcel var1, int var2);

    public interface Creator<T> {
        T createFromParcel(Parcel var1);

        T[] newArray(int var1);
    }

    public interface ClassLoaderCreator<T> extends Parcelable.Creator<T> {
        T createFromParcel(Parcel var1, ClassLoader var2);
    }
}
```

Parcelable 객체를 Detail 화면에서 다시 뽑아내는 과정은 아래와 같다.

```
// Intent 객체를 get 하는 메소드 호출
Intent intent = getIntent();

// 우리는 Movie 객체를 원한다 ! Parcel로부터 Unparcelling 할 때는 내가
// Main 화면에서 정의한 KEY값을 넘겨준다.
Movie movie = intent.getParcelableExtra(PARCEL_KEY);
```

8-3. Implementing Callback functions

1) callback method를 정의한 인터페이스를 구현한다.

```
interface ClickEventHandler{
    public void handleClick();
    //callback 은 이 메소드로 처리해 줄 거야 - 약속만
}
```

2) callback 을 받아서 처리할 Handler를 구현한다.

```
class ClickHandler implements ClickEventHandler{
    public void handleClick(){
      System.out.println("Clicked");
```

```
    //callback 을 받아서 구체적으로 이렇게 처리해 줄거야 - 실제 구현
    }
}
```

3) Click 이벤트의 대상이 되는 (callback을 발생시킬) 클래스를 작성한다.

```
class Button{
    public void onClick(ClickEventHander clickEventHandler){
      clickHandler.handleClick();
      // 버튼 객체에서 onClick 함수를 호출하면 위에서 구현된 메소드가 작동한다.
    }
}
```

4) Button 객체를 만들고 onClick 메소드를 호출하는데, 이 때 인자값으로 ClickEventHandler type 의 reference 를 넘겨준다.

```
public class TestButton{

    public static void main(String [] args){
        Button button = new Button();
        ClickHandler clickHandler = new ClickHandler();

        button.onClick(clickHandler);

        //android 에서는 아래와 같이 인터페이스를 구현하는 익명 클래스를 인자값으로
        //넘겨주는 방식을 주로 취한다.
        Button anonyButton = new Button();
        anonyButton.onClick(new ClickEventHandler{
            @Override
            public void handleClick(){
              System.out.println("Button is Clicked");
            }
        });
    }
}
```

9. Inheritance

상속의 목적 : 코드의 재사용(reusability)

상속의 효과 : subclass 의 기능 확장, 구체화

9-1. Example of Inheritance

```
public class Teacher {
   String designation = "Teacher"; //직함
   String collegeName = "Hanyang";
   void does(){
      System.out.println("Teaching");
   }

  protected void setDesignation(String designation) {
    this.designation = designation;
   }

  protected String getCollegeName() {
    return collegeName;
   }

  protected void setCollegeName(String collegeName) {
    this.collegeName = collegeName;
   }
}
```

Teacher class 를 상속받은 Physics Teacher 를 구현해 보자

```
public class PhysicsTeacher extends Teacher{

  public static void main(String [] args){
      PhysicsTeacher pt = new PhysicsTeacher();
      System.out.println(pt.does()); // superclass 의 does 메소드에 접근 가능
      System.out.println(pt.getDesignation());
      // superclass 의 protected 메소드에 접근 가능
  }

}
```

9-2. super keyword

super 키워드를 통해서 superclass의 메소드에 접근 가능.

```
public class Parent{
  public Parent(){
    System.out.println("Constructor from superclass running...");
  }
  public void invokeSuper(){
    System.out.println("invoking method from superclass");
  }
}
```

위의 Parent class 를 상속받은 child class 를 보자

```
public class Child extends Parent{

  public Child(){
    super(); //명시하지 않아도 superclass 의 생성자를 호출하는 super()가 자동 생성됨
    System.out.println("Constructor from subclass running...");
  }

  @Override
  public void invokeSuper(){
    super.invokeSuper();
    System.out.println("invoking method from the child class");
  }

  public static void main(String [] args){
    Child child = new Child(); // 상속받았으므로, superclass 의 생성자 호출됨
    //그 다음에 subclass 의 생성자 호출
    child.invokeSuper();
    //subclass 의 메소드에서 super class 의 메소드 호출
  }
}
```

9-3. 안드로이드의 MainActivity

안드로이드 스투디오 프로젝트를 처음 만들면 가장 먼저 보게되는 Java 코드이다.

AppCompatActivity 또한 여러 클래스로부터 상속을 받았다.

MainActivity 클래스 안에 커서를 올려놓고 `ctrl + o` 를 클릭하면 창이 하나 뜨는데, 그 창 안에 정의된 메소드들이 우리가 overriding 가능한 메소드 들이다.

```
public class MainActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState); //가장 먼저 superclass 의 onCreate
//함수를 호출하고 있다.
        setContentView(R.layout.activity_main);

    }
}
```

cf. 안드로이드 앱 life cycle → superclass 의 메소드를 override 해서 확인해 보기

10. Abstract class and Abstract methods

10-1. BaseActivity.class

```
public abstract class BaseActivity extends AppCompatActivity{

        @Override
        protected void onCreate(Bundle savedInstanceState) {
          super.onCreate(savedInstanceState);
          setContentView(myView());
        }

        public abstract void checkNetworkConnection();
}
```

MainActivity 가 AppCompatActivity를 상속받은 BaseActivity를 상속받도록 바꿔준다.

이제 BaseActivity를 상속받은 액티비티 클래스들에서 네트워크 연결을 확인하는 메소드를 간편하게 이용할 수 있다.

```
public class MainActivity extends BaseActivity{

    //onCreate 함수 Override 생략

    @Override
    public void checkNetworkConnection(){
      //implement code to check network connection
    };

}
```