10 COMPUTER PROGRAMMING

CLASS AND OBJECT

CONTENTS

- GENERAL STRUCTURE OF THE CLASS
- DECLARATION OF THE CLASS
- OBJECT CREATION
- MEMBER VARIABLE, HOW TO ACCESS MEMBER VARIABLE
- CONSTRUCTOR
- KEYWORD
- METHOD

GENERAL STRUCTURE OF THE CLASS

- **Class is a kind of template to create objects**
- **Java program is a set of Class**
- **Class is composed of 2 main elements**
  - Data property, including data, that an object can have
  - Method that controls data

GENERAL STRUCTURE OF THE CLASS

```
class Class-name  {    //class header

    type1 varName1 = value1;
       ........
    typeN varNameN = valueN;

    Class-name(args1) {
       ..........
    }

    Class-name(argsN) {
       ......
     }

     mtype mName1(margs1)  {
       ......
     }

     mtype mNameN(margsN)  {
   ......
  }
}
```

member variables

constructor

method part

CLASS DECLARTATION

DECLARATION STRUCTURE OF A CLASS

```
[public/final/abstract] class Class-name {   //class header part
  .....    // class member part
}
```

When creating a class, it uses qualifier that specifies the nature of the class.

- public
- final
- abstract

CLASS DECLARTATION

Relation between java program and class

- Principle to define only one class of a program
- Using PUBLIC qualifier in a class with main( ) method
- In case not to specify a qualifier in all classes

CLASS DECLARTATION- example

class declared with only data property

class with data property and method of data

CLASS DECLARTATION- example

Class with data property and Constructor, method

OBJECT CREATION

- Object declaration

- Object creation

- Declaration and creation of the object at the same time

OBJECT CREATION - example

```
class Box {
    int width;
    int height;
    int depth;
}

class MyBox {
    ............
    Box mybox1;
    Box mybox2;
    mybox1 = new Box();
    mybox2 = new Box();
    ............
}
```

Declaration and Creation of the OBJECT

- Object declaration : only refers to the variable with a null value

- Object creation : allocating memory for the object, and then a variable(object_Ref_Var) is having a reference(address) on the object

Object Creation

```
class Box {
  int width;
  int height;
  int depth;
}

class TwoBox {
  public static void main(String args[]) {
    Box mybox1 = new Box();
    Box mybox2 = new Box();
    int vol1, vol2;

    mybox1.width = 20;
    mybox1.height = 40;
    mybox1.depth = 15;

    mybox2.width = 10;
    mybox2.height = 20 ;
    mybox2.depth = 30;

    vol1 = mybox1.width * mybox1.height * mybox1.depth;
    System.out.println(" 첫번째 박스의 부피는 " + vol1 + "입니다.");

    vol2 = mybox2.width * mybox2.height * mybox2.depth;
    System.out.println(" 두번째 박스의 부피는  " + vol2 + "입니다.");
  }
}
```

Member Variable

- Variables declared outside of method within a class
- Use to represent properties that objects can have.
- Classification of member variables

```
[public/private/protected] [static/final]  variable_type   variable_name ;
```

- Declaration for a member variable
  - static : class variable
  - final : final variable
  - Variable not to stick static and final variables :

Member variable - Object variable
(object reference variable and object property variable)

- **Object variable**
  - Representing the attributes that object can have

- **Classification by the value that an object variable represents**
  - Object property variable
  - Object reference variable

Member variable - Object variable
(object reference variable and object property variable)

```
class Box {
  int width;    // object property variable
  int height;   // object property variable
  int depth;    // object property variable
}
class MyBox {
  int vol;
  Box mybox1;
  Box mybox2;
  String boxname;   //

  mybox1 = new Box();
  mybox2 = new Box();
  .....
 }
```

Member variable - Object variable (variable assign')
(object reference variable and object property variable)

- Object property variable : propagated values for a variable is copied

```
.....
int my_count1 = 100;
int my_count2 = my_count1;
        // 객체 속성변수의 대입
Box mybox1 = new Box();
Box mybox2 = mybox1;
        // 객체 참조변수의 대입
.....
```

- Object reference variable: propagated address for a variable is copied, finally, indicating same object to the object

Member variable - Object variable
(object reference variable and object property variable)

```
class Fruit {
  int apple = 5;   // 객체 속성 변수
  int straw = 10;
  int grapes = 15;
}

class Buy extends Fruit {
  public static void main(String[] args) {
        int quantity1, quantity2;

        Fruit f1 = new Fruit();
        Fruit f2 = f1;
        quantity1 = f1.apple + f1.straw + f1.grapes;
        quantity2 = f2.apple + f2.straw + f2.grapes;
        System.out.println("객체 f1의 초기 과일 개수 "+quantity1+"개");
        System.out.println("객체 f2의 초기 과일 개수 "+quantity2+"개");
        f1.apple = 10;
        f2.straw = 20;
        f1.grapes = 30;
        quantity1 = f1.apple + f1.straw + f1.grapes;
        quantity2 = f2.apple + f2.straw + f2.grapes;
        System.out.println("객체 f1의 값 변동 후 개수 "+quantity1+"개");
        System.out.println("객체 f2의 값 변동 후 개수 "+quantity2+"개");
  }
}
```

Member variable – Class variable

- **Declaration format for member variable**

- **Declaration with static**
- **Concept for global variable**
- **The purpose of the class variable**
  - Object variable (object reference, object property) is created whenever objects are created.
  - class variable is created only one, regardless of the number of objects is generated from the class
  - All objects created by one class are sharing class variable.
  -

Member variable – Class variable

Member variable – Class variable

```
class Box {
  int width;
  int height;
  int depth;
  long idNum;
  static  long  boxID = 0;
  public Box() {
    idNum = boxID++;
  }
}

class StaticDemo {
  public static void main(String args[]) {
    Box mybox1 = new Box();
    Box mybox2 = new Box();
    Box mybox3 = new Box();
    Box mybox4 = new Box();
    System.out.println("mybox1의 id 번호 : " + mybox1.idNum);
    System.out.println("mybox2의 id 번호 : " + mybox2.idNum);
    System.out.println("mybox3의 id 번호 : " + mybox3.idNum);
    System.out.println("mybox4의 id 번호 : " + mybox4.idNum);
    System.out.println("전체 박스의 개수는 "+ Box.boxID + "입니다.");
  }
}
```

Member Variable – Final variable

- **Declaration format for Member variable**

```
[public/private/protected] [static/final]  variable_type   variable_name ;
```

- **Using resered keyword Final, specify final variable.**

- **unchangeable constant value**
- **Final variable customarily capitalized**
  - final int MAX = 100;
  - final int MIN = 1;

Member Variable – Access method

- using "." for class variable and object property variable
- class variable
  - class_name.class_variable
- object property variable
  - object_name.object _property_variable

```
class A {
  int aa;
  int bb;
  int cc;
  static int s = 0;
}
class ATest  {
  public static void main(String args[])  {
    A obja = new A();
    obja.aa = 4;                //
    obja.bb = obja.aa * 2;      //
    obja.cc = A.s;              //
    .....................
```

Member Variable Qualifier

- Java uses qualifier to member variable to provide encapsulation and information hiding, encapsulation feature is one of OO features.

- Member Qualifier :
- If a member variable has no any qualifier, the variable is possible to be used in the same package and subclass

Member Variable Qualifier - public

- if a qualifier of a member variable is declared as "public", the variable that belongs to a class which is possible to access can be used.

Member Variable Qualifier - private

- if a qualifier of a member variable is declared as "private", the variable that belongs to a class can be accessed only in the class.

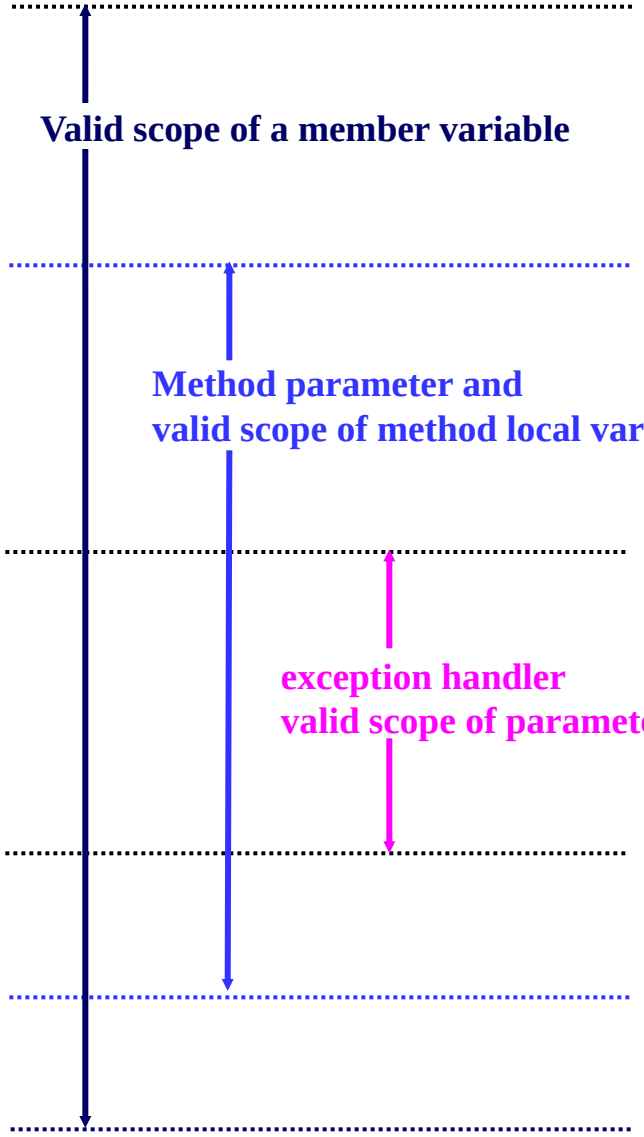
Member Variable Qualifier - protected

- if a qualifier of a member variable is declared as "protected", the variable that belongs to a class can be accessed just through subclass of the class and other classes of the package.

Variable Scope

- **Meaning the region that a variable can be used**
- **Classifying variables based on valid scope**
  - Member variable
  - Method parameter and local variable
  - Exception handler parameter

Variable Scope




```
class MyClass {
    ..............
     member varialbes
    ..............
    public void aMethod(method parameter) {
        ........
          method local varialbes
        ........
        catch(exception parameter) {
            ............
            ............
            ............
        }
        ..........
    }
..............
}
```

CONSTRUCTOR

- Special method describing initializing process, when an object is created from the class.
- The name of the constructor is same as the class name.
- Automatically only one time processing as creating an object
- automatically executed by the NEW operator
- Constructor format

CONSTRUCTOR

Constructor Overloading

- A class is possible to have more then one constructor.
- It is possible to use more than one constructor by overloading in a class.
- If using several constructors, the types and numbers of the parameters should be different.

Constructor Overloading

```
class Box {
  int width;
  int height;
  int depth;
  public Box()  {
    width=1;
    height=1;
    depth=1;
  }
 public Box(int w) {
   width=w;
   height=1;
   depth=1;
 }
  public Box(int w, int h) {
    width=w;
    height=h;
    depth=1;
  }
  public Box(int w, int h, int d) {
    width=w;
    height=h;
    depth=d;
}
```

Constructor Overloading

```
class Box {
  int width, height, depth;
  double dwidth, dheight, ddepth;
  public Box(int w, int h, int d)  {
    width=w;
    height=h;
    depth=d;
  }
 public Box(double w, double h, double d)  {
    dwidth=w;
    dheight=h;
    ddepth=d;
  }
}
```

Reserved word - this

- "this" means a present object.
- Using "this" as parameters of constructor and method, and object variable have same name
- Using "this" calling another constructor in the same class

```
class Box {
  int width;
  int height;
  int depth;
  public void Box(int width, int height, int depth) {
    width=width;
    height=height;
    depth=depth;
  }
}
```

Reserved word – this (EX)

- Use “this” as a name of object property variable and constructor parameter

```
class Box {
  int width;
  int height;
  int depth;
  public Box(int width, int height, int depth) {
    this.width=width;
    this.height=height;
    this.depth=depth;
  }
}
```

Reserved word - this

- Calling another constructor in the same class

```
class Box {
  int width;
  int height;
  int depth;
  public Box() {
    this(1,1,1);
  }
  public Box(int w) {
    this(w,1,1);
  }
  public Box(int w, int h) {
    this(w,h,1);
  }
  public Box(int w, int h, int d)   {
    width=w;
    height=h;
    depth=d;
  }
}
```

Reserved word - this

```
class Sales {
    String title;
    int quantity;
    public Sales(String t){
        this(t,0);
    }
    public Sales(String t, int q){
        title=t;
        quantity=q;
    }
}

class SalesDemo {
     public static void main(String[] args) {
          Sales s1=new Sales("장갑");
          Sales s2=new Sales("양말",200);
          System.out.println("판매품목 :" +s1.title +" " +"수량 :"+s1.quantity);
          System.out.println("판매품목 :" +s2.title +" " +"수량 :"+s2.quantity);
       }
}
```

METHOD

Method

- Define behavior of object that can do
- Generally start with lowercase letter for a method name

- qualifier(public/private/protected) : same meaning of member variable qualifier
- static : class method
- final: final method
- abstract : abstract method
- synchronized : method for synchronizing threads
- Type of returned value : type of returned value after method processed

METHOD

```
class Fruit {
  int apple;
  int straw;
  int grapes;
  int sum;
  Fruit(int apple, int straw, int grapes) {
    this.apple = apple ;
    this.straw = straw ;
    this.grapes = grapes ;
    }
  public int count() {
    sum = apple + straw + grapes;
    return sum;
  }
}
class MethodDemo1  {
  public static void main(String[] args) {
    int total;
    Fruit f1 = new Fruit(30, 30, 30);
    total = f1.count();
    System.out.println("객체 f1의 총 개수 = " + total);
    System.out.println("객체 f1의 apple 개수 = " + f1.apple);
    System.out.println("객체 f1의 straw 개수 = " + f1.straw);
    System.out.println("객체 f1의 grapes 개수 = " + f1.grapes);
  }
}
```

METHOD

```
class Fruit {
  private int a;
  private int s;
  private int g;
  private int sum;
  Fruit(int apple, int straw, int grapes) {
     a = apple ;
     s = straw ;
     g = grapes ;
     this.count();
  }
  private void count() {
     sum = a + s + g;
  }
  public int gettotal() {
       return sum;
  }
  public int getapple() {
       return a;
  }
  public int getstraw() {
     return s;
  }
  public int getgrapes() {
     return g;
  }
}
```

METHOD- (Class Method : static)

- Declaration class method with “static”
- Access using class name like class variable
- Use only class variable within class method

METHOD

```
class Box {
  int width;
  int height;
  int depth;
  long idNum;
  static long boxID = 100;
  static long getcurrentID() {
  return boxID++;
  }
}

class StaticMethodDemo {
  public static void main(String args[]) {
    Box mybox1 = new Box();
    mybox1.idNum = Box.getcurrentID();
    Box mybox2 = new Box();
    mybox2.idNum = Box.getcurrentID();
    System.out.println("mybox1의 id 번호 : " + mybox1.idNum);
    System.out.println("mybox2의 id 번호 : " + mybox2.idNum);
    System.out.println("다음 박스의 번호는 "+ Box.boxID + "번 입니다.");
  }
}
```

METHOD

```
class One{
    int value;
    public One(){
        this(100);
    }
    public One(int value){
       this.value = value;
       Another.methodA(this);
    }
}
class Another{
    static void methodA(One ins){
         System.out.println("메소드A에서의 값: " + ins.value);
    }
}

class OneTest{
    public static void main(String args[]){
         One t1 = new One();
         System.out.println("기본 값: " + t1.value);
         int value = Integer.parseInt(args[0]);
         One t2 = new One(value);
         System.out.println("사용자가 입력한 값: " + t2.value);
    }
}
```

METHOD - final, abstract, synchronized methods

- **final**
  - Class not overriding in sub-class
- **abstract**
  - Declaration within an abstract class
- **synchronized**
  - Method for synchronizing threads

METHOD – method access

- **Format for accessing class method**
  - className.classMethodName(parameter)
- **Format for accessing general method**
  - objectName.objectMethodName(parameter)

METHOD – method return value

- Specifying return value in the part of method declaration
- "void" in case of no return value
- Return values of reference data type as well as primitive data type

```
public int sum(int a, int b) {
  int c;
  c = a + b;
  return c;
}
```

```
public Box volume_compute(Box instance_box) {
  Box v_box = new Box();
  v_box.width = instance_box.width;
  v_box.height = instance_box.height;
  v_box.depth = instance_box.depth;
  v_box.volume= v_box.width * v_box.height * v_box.depth;
  return v_box;
}
```

METHOD OVERLOADING

- Same concept like constructor overloading
- Use same method name within a class
- The methods of the same name must be different in type or number of parameters.
- Polymorphism
  - Performing various operations in one method name

METHOD OVERLOADING

```
Class Overload1{
void test() {
    System.out.println("매개변수 없음");
  }
 void test(int a) {
    System.out.println("매개변수 int " + a);
  }
 void test(int a, int b) {
    System.out.println("매개변수 int " + a + "와 int " + b);
  }
  double test(double a) {
    System.out.println("매개변수 double " + a);
    return a * 2;
  }
}

class OverloadDemo1 {
  public static void main(String args[]) {
    Overload1 ob = new Overload1();
    double result;

    ob.test();
    ob.test(100);
    ob.test(5, 10);
    result = ob.test(4.2);
    System.out.println("ob.test(4.2)의 결과 : " + result);
  }
}
```

METHOD OVERLOADING

```
class Overload2 {
  void test() {
    System.out.println("매개변수 없음");
  }
  void test(int a, int b) {
    System.out.println("매개변수 int " + a + "와 int " + b);
  }
  void test(double a) {
    System.out.println("매개변수 double " + a);
  }
}
class OverloadDemo2 {
  public static void main(String args[]) {
    Overload2 ob = new Overload2();
    int i = 88;
    ob.test();
    ob.test(10, 20);
    ob.test(i);
    ob.test(123.2);
  }
}
```

METHOD OVERLOADING

```
class OverloadDemo3 {
    public static void main(String args[]){
    Overload ol = new Overload();
        int input[] = new int[args.length];
        for(int i=0; i<args.length; i++)
            input[i] = Integer.parseInt(args[i]);
            switch (args.length){
               case 0:
                  ol.calc();
                  break;
               case 1:
                   ol.calc(input[0]);
                   break;
               case 2:
                   ol.calc(input[0], input[1]);
                   break;
              case 3:
                   ol.calc(input[0], input[1], input[2]);
                   break;
               default:
                   System.out.println("인수의 개수가 많습니다.");
               }
    }
}
```

```
class Overload{
  void calc(){
        System.out.println("매개 변수가 없습니다.");
   }
   void calc(int width){
        System.out.println("정사각형의 넓이" + width * width);
  }
   void calc(int width, int height){
        System.out.println("직사각형의 넓이" + width * height);
   }
   void calc(int width, int height, int depth){
       System.out.println("직육면체의 부피" + width * height *
         depth);
   }
}
```

Argument passing to method

- **call by value : as delivering primitive data type**
  - Copy actual parameter value to formal parameter
  - No changing in actual parameter although formal parameter is changing
- **call by reference : as delivering reference data type**
  - Delivering addresses of actual parameters(object) to formal parameters
  - If formal parameter value is changing, the actual parameter should be changed

Argument passing to method

```
class ParaPassing1 {
  public void change(int i, int j[], StringBuffer sb) {
    i = 20;
    j[3] = 400;
    sb.append(" 알기쉽게 해설한 『자바』 랍니다.");
  }

  public void display(int i, int j[], StringBuffer sb) {
    System.out.println("객체 속성 변수 i의 값 : " + i);
    System.out.print("배열의 값 : ");
    for(int index = 0; index < j.length; index++)
      System.out.print(j[index] + " ");
    System.out.println("");
    System.out.println("문자열 sb의 값 : " + sb);
  }
}
```

```
class ParaPassingDemo1 {

  public static void main(String args[]) {
    ParaPassing1 cb = new ParaPassing1();
    int i = 10;
    int j[]  = { 1, 2, 3, 4 };
    StringBuffer sb = new StringBuffer("배우기 쉽죠?");
    System.out.println("첫번째 display() 메소드 호출");
    cb.display(i, j, sb);
    cb.change(i, j, sb);
    System.out.println("============================");
    System.out.println("값을 변환한 다음 두번째 display() 호출");
    cb.display(i, j, sb);
  }
}
```

finalize method and garbage collection

- JVM(java virtual machine) automatically processes garbage collection to objects that do not be needed for operating efficient system.
- JVM calls finalize( ) method of an object before collecting garbage.
- Using finalize() method as creating a class, user describes the processes of returning resources like file or socket that use in object.

CONCLUDE

- GENERAL STRUCTURE OF THE CLASS
- DECLARATION OF THE CLASS
- OBJECT CREATION
- MEMBER VARIABLE, HOW TO ACCESS MEMBER VARIABLE
- CONSTRUCTOR
- KEYWORD
- METHOD