

# 5. Defining Classes and Methods

[ITP20003] Java Programming

# Agenda

---



- Class and Method Definitions
- Information Hiding and Encapsulation
- **Objects and References**

# Variables of a Class Type

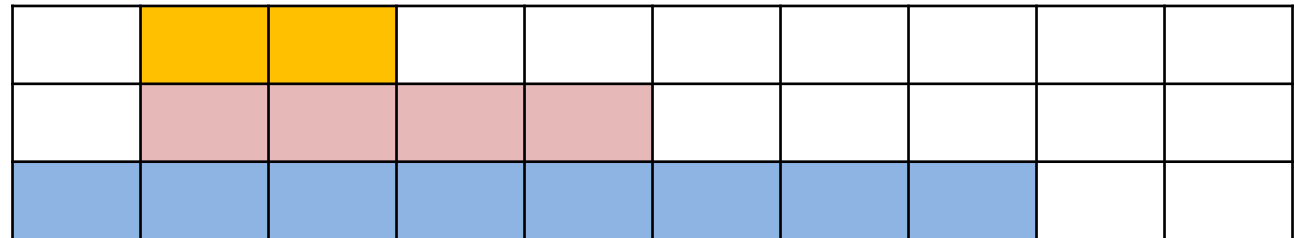
- All variables are implemented as a memory location

In memory

char c

int number

double number2



- Variable of *primitive type* contains **data** in the memory location assigned to the variable

Ex) int i;

- Variable of *class type* contains **memory address of object** named by the variable

Ex) MyClass obj = new MyClass();

# Variables of a Class Type

---



- Object itself not stored in the variable
  - Stored elsewhere in memory
  - Variable contains **address** of where it is stored
- Address is called the *reference* to the variable
- A *reference type variable* holds references (memory addresses)
  - This makes memory management of class types more efficient

# Variables of a Class Type

```
public class Student_ver2 {
    private String name;
    private int score;
    private String grade;

    public void writeoutput() {
        String grade;
        if (score > 50)
            grade = "pass";
        else
            grade = "fail";
        System.out.println(name + ": " + score + ": " + grade);
    }

    public int getScore() {return score;}
    public String getName(){return name;}

    public void setdata(String s_name, int s_score){
        name = s_name;
        score = s_score;
    }
}
```

# Variables of a Class Type

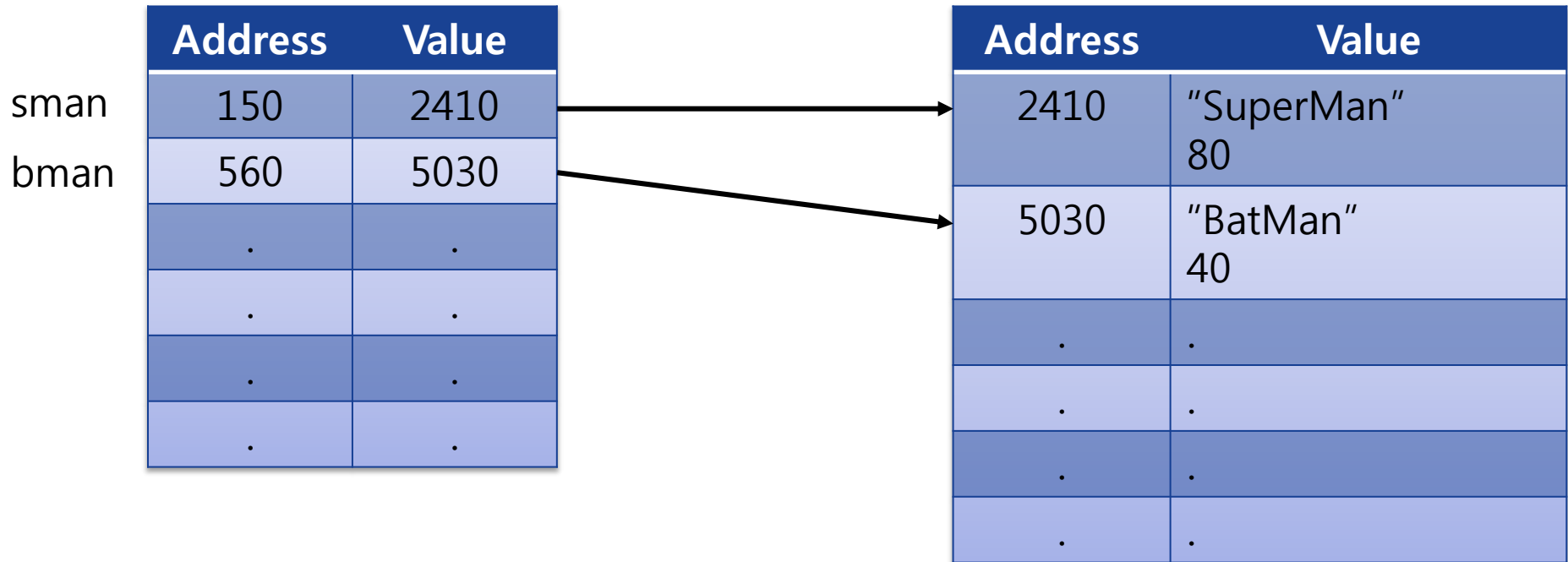


## Student\_main\_ver3.java

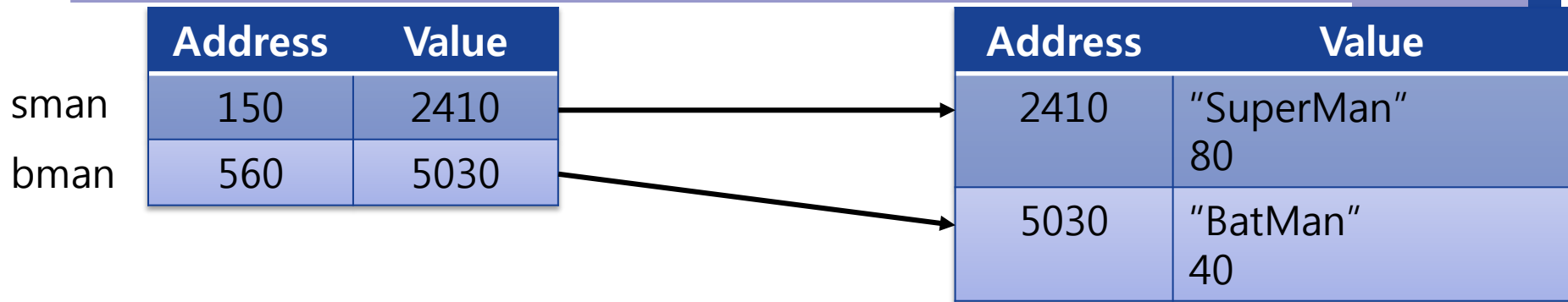
```
public class Student_main_ver3 {  
    public static void main(String[] args) {  
        Student_ver2 sman = new Student_ver2();  
        Student_ver2 bman = new Student_ver2();  
        sman.setdata("SuperMan", 80);  
        bman.setdata("BatMan", 40);  
        sman.writeoutput();  
        bman.writeoutput();  
    }  
}
```

**SuperMan: 80: pass**  
**BatMan: 40: fail**

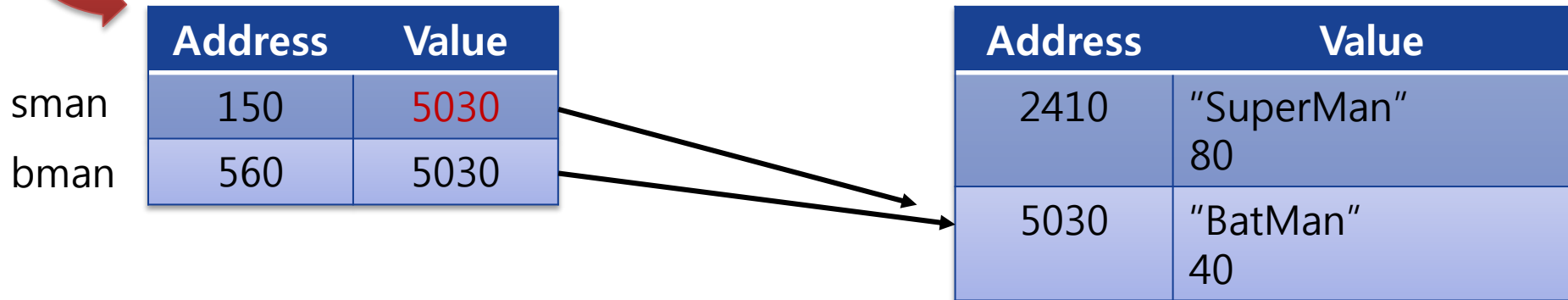
# Variables of a Class Type



# Variables of a Class Type



sman=bman;



Be careful!!

'sman=bman' statement will change the value at address 150 to the value at the address 560.



# Defining an equals Method

- Remember variables of a class type contain memory addresses (NOT objects themselves).  
Then what would '==' return?

Student\_main\_ver4.java

```
public class Student_main_ver4 {  
    public static void main(String[] args) {  
        Student_ver2 sman = new Student_ver2();  
        Student_ver2 sman2 = new Student_ver2();  
        sman.setdata("SuperMan", 80);  
        sman2.setdata("SuperMan", 80);  
        sman.writeoutput();  
        sman2.writeoutput();  
  
        if (sman==sman2)  
            System.out.print(">> Same!!");  
        else  
            System.out.print(">> Not same!!");  
    }  
}
```

# Defining an equals Method

```
SuperMan: 80: pass  
SuperMan: 80: pass  
>> Not same!!
```

- They are not same.
- Because '*smán*' and '*smán2*' contain the addresses of the objects.
- To compare two objects, we need *better strategy*.

# Defining an equals Method

Student_ver2	Student_ver3
- name: String - score: int	- name: String - score: int
+ writeoutput(): void + getName(): String + getScore(); int + setdata(String s_name, int s_score): void	+ writeoutput(): void + getName(): String + getScore(); int + setdata(String s_name, int s_score): void <b>+ equals(Student_ver3 in_object): boolean</b>

```
public boolean equals(Student_ver3 in_object)
{
    boolean b;
    b = (this.name.equalsIgnoreCase(in_object.name) &&
        this.score == in_object.score);

    return b;
}
```

# Defining an equals Method



```
public class Student_main_ver5 {  
    public static void main(String[] args) {  
        Student_ver3 sman = new Student_ver3();  
        Student_ver3 sman2 = new Student_ver3();  
        sman.setdata("SuperMan", 80);  
        sman2.setdata("SuperMan", 80);  
        sman.writeoutput();  
        sman2.writeoutput();  
  
        if (sman.equals(sman2))  
            System.out.print(">> Same!!");  
        else  
            System.out.print(">> Not same!!");  
    }  
}
```

SuperMan: 80: pass  
SuperMan: 80: pass  
>> Same!!

# .equals()



- Every class has a default *.equals()* method if it is not explicitly written
  - Does not necessarily do what you want.
- You decide what it means for two objects of a specific class type to be considered *equal*.
  - Perhaps Students are equal if the names and student numbers are equal.
  - Put this logic inside *.equals()* method.

# Parameters of a Class Type



- When **assignment operator** used with objects of class type
  - **Only memory address is copied**
- Similar to use of parameter of class type
  - Memory address of actual parameter passed to formal parameter
  - Formal parameter may access public elements of the class
    - Actual parameter thus can be changed by class methods

# Parameters of a Class Type

- Java passes arguments to a method. Remember that we called a method 'setdata()' and provided **parameters**.

```
public void setdata(String s_name, int s_score){  
    name          = s_name;  
    score          = s_score;  
}
```

- For primitive type, the parameter contains the value of its corresponding argument.
  - Call-by-value
  - It is impossible to change the original data.
- For class type, the reference (address) to the class object is passed to the parameters.
  - Call-by-reference.
  - It is possible to change the data in an object.

# Student\_main\_ver6.java



```
public class Student_main_ver6 {
    public static void main(String[] args) {
        int class_score = 80;
        Student_ver3 sman = new Student_ver3();
        sman.setdata("SuperMan", class_score);

        System.out.println("\n\nBefore");
        System.out.println("class_score is " + class_score );
        sman.writeoutput();

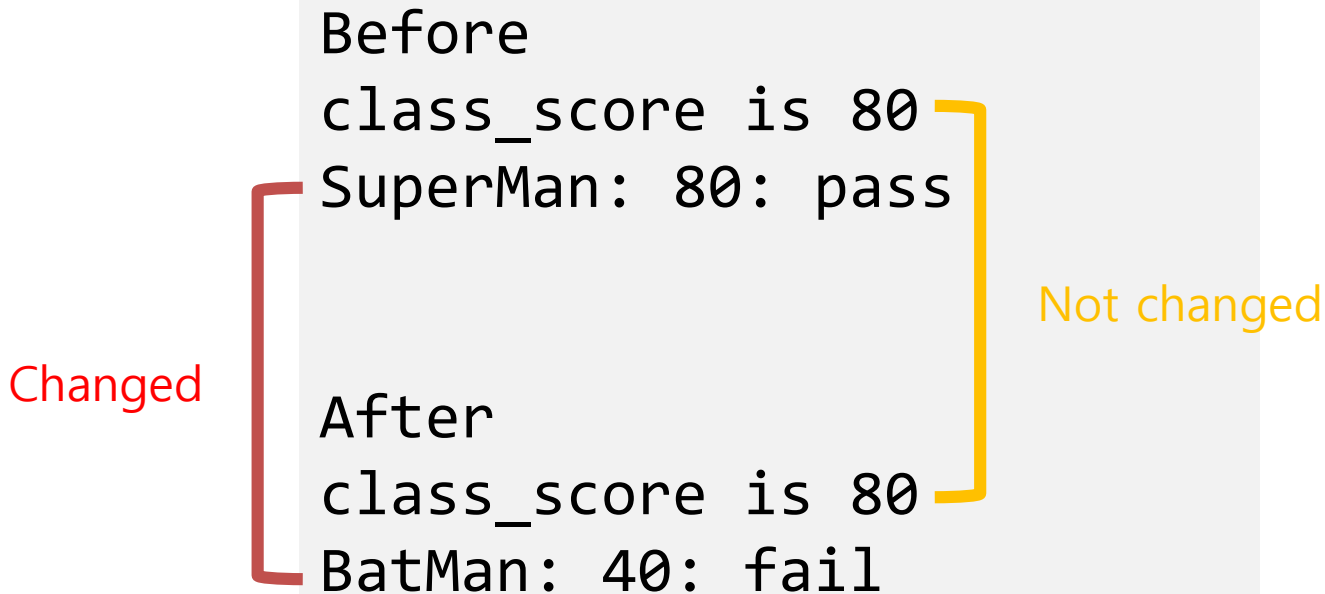
        changetest(sman, class_score);

        System.out.println("\n\nAfter");
        System.out.println("class_score is " + class_score );
        sman.writeoutput();
    }

    public static void changetest(Student_ver3 in_object, int class_score){
        int new_score = 40;
        in_object.setdata("BatMan", new_score);
        class_score = new_score;
    }
}
```



# Parameters of a Class Type



Before

```
class_score is 80  
SuperMan: 80: pass
```

Changed

After

```
class_score is 80  
BatMan: 40: fail
```

Not changed

# 6. More About Objects and Methods

[ITP20003] Java Programming

# Agenda

---



- **Constructors**
- Static Variables and Static Methods
- ~~Writing Methods~~
- Overloading
- Information Hiding Revisited
- Enumeration As A Class
- ~~Packages~~

# Defining Constructors



- A special method is called when instance of an object created with *new*
  - Create objects
  - Initialize values of instance variables
- Can have parameters
  - To specify initial values if desired
- May have multiple definitions with different numbers or types of parameters

# Brief version of Pet example



```
public class Pet{
    private String name;
    private int age; //in years
    private double weight;//in pounds

    public Pet(String initialName, int initialAge, double initialWeight){
        name = initialName;
        if ((initialAge < 0) || (initialWeight < 0)){
            System.out.println("Error: Negative age or weight.");
            System.exit(0);
        }
        else{
            age = initialAge;
            weight = initialWeight;
        }
    }

    public String getName()           {return name;}
    public int getAge()               {return age;}
    public double getWeight()         {return weight;}
    public void writeOutput(){
        System.out.println("Name: " + name);
        System.out.println("Age: " + age + " years");
        System.out.println("Weight: " + weight + " pounds");
    }
}
```

# Brief version of Pet example

```
public class Pet_main {  
    public static void main(String[] args) {  
        Pet p1 = new Pet("cat",5,2.2);  
        Pet p2 = new Pet("dog",10,12.6);  
  
        p1.writeOutput();  
        p2.writeOutput();  
    }  
}
```

Name: cat

Age: 5 years

Weight: 2.2 pounds

Name: dog

Age: 10 years

Weight: 12.6 pounds

# Defining Constructors



- Constructor without parameters is the default constructor
  - Java will define this automatically if the class designer does not define **ANY** constructors
  - If you do define a constructor, Java will not automatically define a default constructor
- Usually default constructors not included in class diagram

# Constructor

```
public class Pet{
    private String name;
    private int age; //in years
    private double weight;//in pounds

    public Pet(String initialName, int initialAge, double initialWeight){
        . . .
    }

    public String getName()           {return name;}
    public int getAge()                {return age;}
    public double getWeight()          {return weight;}
    public void writeOutput(){
        System.out.println("Name: " + name);
        System.out.println("Age: " + age + " years");
        System.out.println("Weight: " + weight + " pounds");
    }
}
```



# Constructor

```
public class Pet_main {  
    public static void main(String[] args ) {  
        Pet p1 = new Pet();  
        p1.set( "cat" ,5,2.2);  
  
        Pet p2 = new Pet();  
        p2.set( "dog" ,10,12.6);  
  
        p1.writeOutput();  
        p2.writeOutput();  
    }  
}
```

```
public class Pet_main {  
    public static void main(String[] args ) {  
        Pet p1 = new Pet( "cat" ,5,2.2 );  
        Pet p2 = new Pet( "dog" ,10,12.6 );  
  
        p1.writeOutput();  
        p2.writeOutput();  
    }  
}
```

**Name: cat**  
**Age: 5 years**  
**Weight: 2.2 pounds**  
**Name: dog**  
**Age: 10 years**  
**Weight: 12.6 pounds**

# Constructor vs. Set Method



- Constructors are called only when you create an object . To change the state of an existing object, you need one or more set methods.
- So, if you want to reset the instance variables, you can use 'set method' for it.

# Constructor

---



- Generally, constructor should contain all initialization logic
- assign initial values based on input parameters
- assign default initial values without input
- reserve resource, prepare input/output stream
- whatever other logic necessary (e.g., error checking )

# Default constructor

- A constructor without parameters is called the default constructor. Most of classes you define should include a default constructor.

```
public class Pet
{
    private String name;
    private int age;//inyears
    private double weight;//inpounds
    public Pet(){
        name="Nonameyet.";
        age=0;
        weight=0;
    }
    ...
}
```

# Default constructor

Pet\_default\_constructor.java

```
public class Pet_default_constructor {
    private String name ;
    private int age ; //in years
    private double weight ; //in pounds
    public void writeOutput(){
        System.out.println( "Name: " + name );
        System.out.println( "Age: " + age + " years" );
        System.out.println( "Weight: " + weight + " pounds" );
    }
}
```

Pet\_main\_default\_constructor.java

```
public class Pet_main_default_constructor {
    public static void main(String[] args ) {
        Pet_default_constructor p1 = new Pet_default_constructor();
        p1.writeOutput();
    }
}
```

# Default constructor

---



- What if you did not write any constructor?
- Java make a default constructor for each class if you do not define any constructor .
- It assigns a default value to each instance variable.
  - integer, double: 0
  - String and other class-type variables: null
  - boolean: false
- If you define at least one constructor, a default constructor will not be created for you.

# Multiple constructor

```
public class Pet{
    private String name;
    private int age; //in years
    private double weight;//in pounds

// Initialize name, age, weight
public Pet(String initialName, int initialAge, double initialWeight){. . .}

public Pet(String initialName){. . .}    // Initialize name
public Pet(int initialAge){. . .}        // Initialize age
public Pet(double initialWeight){. . .}  // Initialize weight

public String getName()                  {return name;}
public int getAge()                      {return age;}
public double getWeight()                {return weight;}
public void writeOutput(){
    System.out.println("Name: " + name);
    System.out.println("Age: " + age + " years");
    System.out.println("Weight: " + weight + " pounds");
}}
```

# Agenda

---



- Constructors
- **Static Variables and Static Methods**
- ~~Writing Methods~~
- Overloading
- Information Hiding Revisited
- Enumeration As A Class
- ~~Packages~~



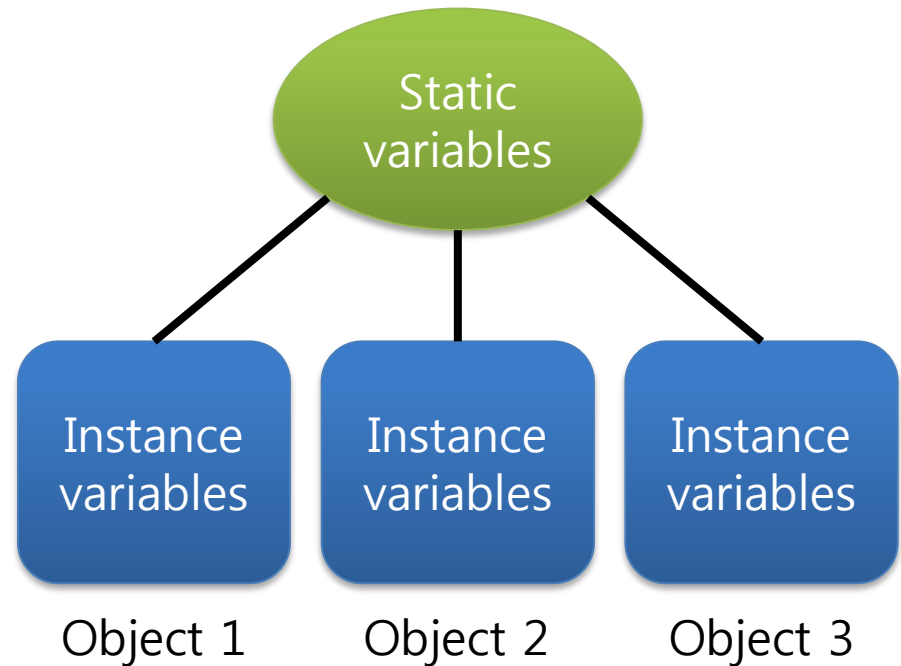
# Static Variables



- **Static variables**, also called **class variables**, are shared by all objects of a class
  - Only one instance of the variable exists
    - Contrast with instance variables
  - Variables declared **static final** are considered **constants** – value cannot be changed
  - Variables declared **static (without final)** can be changed

# Static Variables

- A static variable is shared by all the objects of its class.
- The declaration of a static variable contains the keyword static.
- Three kinds of variables
  - Local variables
  - Instance variables
  - Static (class) variables.



# Static Variables

```
import java.util.Scanner;

public class Main {

    public int instanceVar;           // declared as public only for demonstration
    public static int staticVar;      // declared as public only for demonstration

    public static void main(String args[]){
        System.out.println("MyClass.staticVar = " + Main.staticVar);

        Main a1 = new Main();
        Main a2 = new Main();

        a1.instanceVar=1; a1.staticVar=100;
        a2.instanceVar=2; a2.staticVar=200;

        System.out.println("a1.instanceVar = " + a1.instanceVar);
        System.out.println("a2.instanceVar = " + a2.instanceVar);
        System.out.println("a1.staticVar = " + a1.staticVar);           // also possible
        System.out.println("a2.staticVar = " + a2.staticVar);           // also possible

        a1.instanceVar++;
        a1.staticVar++;
        System.out.println("a1.instanceVar = " + a1.instanceVar);
        System.out.println("a1.staticVar = " + a1.staticVar);
        System.out.println("a2.instanceVar = " + a2.instanceVar);
        System.out.println("a2.staticVar = " + a2.staticVar);
    }
}
```

# Static Variables

---



```
Main.staticVar = 0  
a1.instanceVar = 1  
a2.instanceVar = 2  
a1.staticVar = 200  
a2.staticVar = 200  
a1.instanceVar = 2  
a1.staticVar = 201  
a2.instanceVar = 2  
a2.staticVar = 201
```

# Static Methods



- Some methods may have **no relation to any type of object**  
Ex)
  - Compute max of two integers
  - Convert character from upper- to lower case
- **Static method** declared in a class
  - Can be invoked **without using an object**
  - Instead **use the class name**
  - **Cannot access instance variables or instance methods**

# Static Methods

```
public class DimensionConverter
{
    public static final int INCHES_PER_FOOT = 12;

    public static double convertFeetToInches (double feet)
    {
        return feet * INCHES_PER_FOOT;
    }

    public static double convertInchesToFeet (double inches)
    {
        return inches / INCHES_PER_FOOT;
    }
}
```

# Static Methods

```
import java.util.Scanner;

public class DimensionConverterDemo
{
    public static void main (String [] args)
    {
        Scanner keyboard = new Scanner (System.in);
        System.out.println ("Enter a measurement in inches: ");
        double inches = keyboard.nextDouble ();
        double feet = DimensionConverter.convertInchesToFeet (inches);
        System.out.println (inches + " inches = " + feet + " feet.");
        System.out.print ("Enter a measurement in feet: ");
        feet = keyboard.nextDouble ();
        inches = DimensionConverter.convertFeetToInches (feet);
        System.out.println (feet + " feet = " + inches + " inches.");
    }
}
```

# Static Methods



## ■ Result

```
Enter a measurement in inches: 18
18.0 inches = 1.5 feet.
Enter a measurement in feet: 1.5
1.5 feet = 18.0 inches.
```



# Static Methods

```
public class StaticMethodDemo {
    private int att1, att2;
    public void InstanceMethod() {
        System.out.println("This is an instance method.");
        System.out.println("\tatt1 = " + att1 + ", att2 = " + att2);
    }

    public static void StaticMethod() {
        System.out.println("This is a static method.");
        // System.out.println("\tatt1 = " + att1 + ", att2 = " + att2); // not allowed
    }
}
```

```
public class OtherClass {
    public static void main(String args[]){
        // calling instance method
        StaticMethodDemo a = new StaticMethodDemo();
        a.InstanceMethod();

        // calling static method
        StaticMethodDemo.StaticMethod();
        a.StaticMethod(); // also possible
    }
}
```

# Tasks of *main* in Subtasks



- Program may have complicated logic or repetitive code
- Create **static methods** to accomplish subtasks
  - *main* cannot call instance methods because it's a static method.

Ex)

- Consider [example code](#), listing 6.9  
a main method with repetitive code
- Note [alternative code](#), listing 6.10  
uses helping methods

# The *Math* Class

- Provides many standard mathematical methods
  - Automatically provided, no import needed

Name	Description	Argument Type	Return Type	Example	Value Returned
pow	Power	double	double	Math.pow(2.0, 3.0)	8.0
abs	Absolute value	int, long, float, or double	Same as the type of the argument	Math.abs(-7) Math.abs(7) Math.abs(-3.5)	7 7 3.5
max	Maximum	int, long, float, or double	Same as the type of the arguments	Math.max(5, 6) Math.max(5.5, 5.3)	6 5.5

# The *Math* Class

Name	Description	Argument Type	Return Type	Example	Value Returned
min	Minimum	int, long, float, or double	Same as the type of the arguments	Math.min(5, 6) Math.min(5.5, 5.3)	5 5.3
round	Rounding	float or double	int or long, respectively	Math.round(6.2) Math.round(6.8)	6 7
ceil	Ceiling	double	double	Math.ceil(3.2) Math.ceil(3.9)	4.0 4.0
floor	Floor	double	double	Math.floor(3.2) Math.floor(3.9)	3.0 3.0
sqrt	Square root	double	double	sqrt(4.0)	2.0

# Random Numbers



- `Math.random()` returns a random double that is greater than or equal to zero and less than 1
  - Can scale using addition and multiplication; the following simulates rolling a six sided die

Ex) `int die = (int) (6.0 * Math.random()) + 1;`
- Java also has a `Random` class to generate random numbers

# Wrapper Classes



- Recall that arguments of primitive type treated differently from those of a class type
  - May need to treat primitive value as an object
- Java provides **wrapper classes** for each primitive type  
*Byte, Short, Integer, Float, Double, Character, Boolean*
  - Allow programmer to have an **object** that corresponds to value of primitive type
  - Contain useful **predefined constants and methods**
  - Wrapper classes have no default constructor
  - Wrapper classes have no *set* methods

# Wrapper Classes

- Static methods in class *Character*

Name	Description	Argument Type	Return Type	Examples	Return Value
toUpperCase	Convert to uppercase	char	char	Character.toUpperCase('a') Character.toUpperCase('A')	'A' 'A'
toLowerCase	Convert to lowercase	char	char	Character.toLowerCase('a') Character.toLowerCase('A')	'a' 'a'
isUpperCase	Test for uppercase	char	boolean	Character.isUpperCase('A') Character.isUpperCase('a')	true false

# Wrapper Classes

## ■ Static methods in class *Character*

Name	Description	Argument Type	Return Type	Examples	Return Value
isLowerCase	Test for lowercase	char	boolean	Character.isLowerCase('A') Character.isLowerCase('a')	false true
isLetter	Test for a letter	char	boolean	Character.isLetter('A') Character.isLetter('%')	true false
isDigit	Test for a digit	char	boolean	Character.isDigit('5') Character.isDigit('A')	true false
isWhitespace	Test for whitespace	char	boolean	Character.isWhitespace(' ') Character.isWhitespace('A')	true false
Whitespace characters are those that print as white space, such as the blank, the tab character ('\\t'), and the line-break character ('\\n').					



# Agenda

---



- Constructors
- Static Variables and Static Methods
- ~~Writing Methods~~
- **Overloading**
- Information Hiding Revisited
- Enumeration As A Class
- ~~Packages~~

# Overloading Basics

---



- Two or more methods may have the same name within the same class.
- Java distinguishes the methods by **number and types of parameters**.
  - If it cannot match a call with a definition, it attempts to do type conversions.
  - A method's name and number and type of parameters is called **the signature**.

# Overload

Overload.java



```
public class Overload{
    public static void main(String[] args){
        double average1 = Overload.getAverage(40.0, 50.0);
        double average2 = Overload.getAverage(1.0, 2.0, 3.0);
        char average3     = Overload.getAverage('a', 'c');
        System.out.println("average1 = " + average1);
        System.out.println("average2 = " + average2);
        System.out.println("average3 = " + average3);
    }
    public static double getAverage(double first, double second){
        System.out.println("1st Method");
        return (first + second) / 2.0;
    }
    public static double getAverage(double first, double second, double third){
        System.out.println("2nd method");
        return (first + second + third) / 3.0;
    }
    public static char getAverage(char first, char second){
        System.out.println("3rd method");
        return (char)((((int)first + (int)second) / 2);
    }
}
```

# Overloading and Type Conversion

- Overloading and automatic type conversion can conflict
- Remember the compiler attempts to overload before it does type conversion
  - Ex) The Pet class has two constructors

```
public Pet (int initialAge)
public Pet (double initialWeight)
```

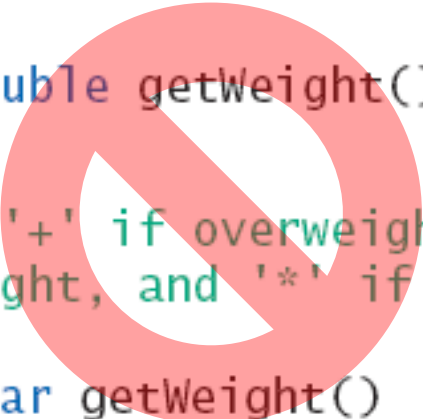
    - If we pass an **integer** to the constructor we get **the constructor for age**, even if we intended the constructor for weight
- Use descriptive method names, avoid overloading

# Overloading and Return Type

- You must not overload a method where the only difference is the type of value returned
  - The signatures are the same

```
/**
 Returns the weight of the pet.
 */
public double getWeight()

/**
 Returns '+' if overweight, '-' if
 underweight, and '*' if weight is OK.
 */
public char getWeight()
```



# Automatic type conversion

- A class has the following method,

```
public Pet(int initialAge);  
public Pet(double initialWeight);
```

- If we create a Pet:  
Pet myPet = new Pet(35);
- But we **should've used 35.0**  
if we meant the weight for the pet.
- You'd better to **use explicit (specific & correct) value/variables**  
rather than implicit (automatic) type conversion.

# Agenda

---



- Constructors
- Static Variables and Static Methods
- ~~Writing Methods~~
- Overloading
- **Information Hiding Revisited**
- Enumeration As A Class
- ~~Packages~~

# Information hiding

---



- For information hiding, we can use 'private' for modifier.
- Because access using accessor ('setter') and mutator ('getter') methods makes the information in an object safe and hide.
- Really.....?



# Privacy Leaks

---



- Instance variable of a class type contain address where that object is stored
- Assignment of class variables results in two variables pointing to same object
  - Use of method to change either variable, changes the actual object itself

*Write two classes “Pet.java” and “PetPair.java”*

```
public class Pet
{
    private String name;
    private int age; //in years
    public Pet(String initialName, int initialAge){
        name = initialName;        age = initialAge;
    }
    public void setPet(String initialName, int initialAge){
        name = initialName;        age = initialAge;
    }
    public String getName(){return name;}
    public int getAge(){return age;}
    public void writeOutput(){
        System.out.print("Name: " + name);
        System.out.println("\tAge: " + age + " years");
    }
}
```

```
public class PetPair{
    private Pet first, second;
    public PetPair(Pet firstPet, Pet secondPet){
        first = firstPet;        second = secondPet;
    }
    public Pet getFirst(){return first;}
    public Pet getSecond(){return second;}
    public void writeOutput(){
        System.out.println("First pet in the pair:");
        first.writeOutput();
        System.out.println("\nSecond pet in the pair:");
        second.writeOutput();
    }
}
```

# Privacy leaks

```
public class Petpair_main {
    public static void main(String[] args) {
        Pet goodDog      = new Pet("Dog", 5);
        Pet goodCat      = new Pet("Cat", 4);
        PetPair pair      = new PetPair(goodDog, goodCat);
        System.out.println("Our pair:");
        pair.writeOutput( );
    }
}
```

```
public class Petpair_main_hack {
    public static void main(String[] args) {
        Pet goodDog      = new Pet("Dog", 5);
        Pet goodCat      = new Pet("Cat", 4);
        PetPair pair      = new PetPair(goodDog, goodCat);
        System.out.println("Our pair:");
        pair.writeOutput( );

        Pet badGuy       = pair.getFirst();
        badGuy.setPet("Hacked", 1200);
        System.out.println("\n\nOur pair now:");
        pair.writeOutput( );
    }
}
```

# Privacy leaks



## Petpair\_main.java

```
Our pair:  
First pet in the pair:  
Name: Dog      Age: 5 years  
  
Second pet in the pair:  
Name: Cat      Age: 4 years
```

## Petpair\_main\_hack.java

```
Our pair:  
First pet in the pair:  
Name: Dog      Age: 5 years  
  
Second pet in the pair:  
Name: Cat      Age: 4 years  
  
Our pair now:  
First pet in the pair:  
Name: Hacked Age: 1200 years  
  
Second pet in the pair:  
Name: Cat      Age: 4 years
```

# Privacy leaks



The instance variables in the first object are changed.  
This phenomenon is called a privacy leak.

## Suggestions to avoid a privacy leak.

- Declare the instance variable's type to be of **a class** that has **no set methods**, such as the class String.
- Omit accessor methods that return an object named by an instance variable of a class type. Instead, **define methods that return individual attributes** of such an object.
- Make accessor methods **return a clone of any object** named by an instance variable of a class type, instead of the object itself.

# Enumeration as a Class

- Consider defining an enumeration for suits of cards

```
enum Suit { CLUBS, DIAMONDS, HEARTS, SPADES }
```
- Compiler creates a **class *Suit*** with methods
  - equals() – tests whether current object is the same with other object
  - compareTo() - compares with other Suit object. It returns a negative, zero or a positive according to the comparison result
  - ordinal() returns the position or the ordinal value
  - toString() – returns the string form such as “HEARTS”.
  - valueOf() – eg. Suit.valueOf(“HEARTS”) returns the object Suit.HEARTS.

For more,

<http://docs.oracle.com/javase/1.5.0/docs/api/java/lang/Enum.html>

# Enumeration as a Class



```
enum Suit{  
    // strings as the values for the enumerated objects  
    CLUBS ("black"), DIAMONDS ("red"), HEARTS ("red"), SPADES ("black");  
  
    private final String color;  
    private Suit (String suitColor) {          color = suitColor; }  
    public String getColor ()                  {          return color;      }  
}
```

```
public class Suit_main {  
    public static void main(String[] args) {  
        Suit cardSuit = Suit.DIAMONDS;  
        System.out.println(cardSuit.ordinal());  
        System.out.println(cardSuit.getColor());  
  
        System.out.println("\nAfter changing value: \n");  
  
        cardSuit = Suit.SPADES;  
        System.out.println(cardSuit.ordinal());  
        System.out.println(cardSuit.getColor());  
    }  
}
```

## Output

1  
red

After changing value:

3  
black

# Agenda

---



- Constructors
- Static Variables and Static Methods
- Writing Methods
- Overloading
- Information Hiding Revisited
- Enumeration As A Class
- **Packages**



# Packages

- Remember? We have seen this kind of code.

```
import java.util.Scanner;
public class WhileDemo
{
    public static void main(String[] args)
    {
        int count, number;
        System.out.println("Enter a number");
        Scanner keyboard = new
        Scanner(System.in);
        number = keyboard.nextInt();
        ...
    }
}
```

- We want to divide a big project into multiple components ( at many levels ).

# Packages

---



- A package is a collection of classes grouped together into a folder and given a package name.
- Each class is in a separate file named after the class.
- Project > Package > Class > Method

# Packages



- Each file in the package **must begin with a package statement**, ignoring blank lines and comments.

## SYNTAX FOR A CLASS IN A PACKAGE

<Blank lines or comments.>

```
package Package_Name;
```

<A class definition.>

## EXAMPLES

```
package general.utilities;
```

```
package java.io;
```

# The import Statement

- You can use all the classes that are in a package within any program or class definition by placing an import statement that names the package at the start of the file containing the program or class definition. The program or class need not be in the same folder as the classes in the package.

## SYNTAX

`import Package_Name.Class_Name_Or_Asterisk;`

Writing a class name imports just a single class from the package; writing an \* imports all the classes in the package.

## EXAMPLES

`import java.util.Scanner;`     *=> loads only Scanner class.*

`import java.io.*;`     *=> loads all the classes in the package.*

# Existing Packages in Java



- Java has many packages (java.lang)
- java.lang package includes classes for basic java programming.
- java.lang does not need 'import'.
- java.lang includes
  - Object** class : the parent class of all the classes in java
  - Wrapper** class
  - String** class
  - System** class
  - Math** class
  - etc.

# Existing Packages in Java

---



## **java.util**

classes for representing and manipulating data collection

We have used Scanner, Random, ArrayList in this package

## **java.awt and java.swing**

classes for building graphical user interface

We have used them in lab 3 and lab 5.

## **java.io**

file operations (read / write files)

We will discuss this one soon

# A Package Name



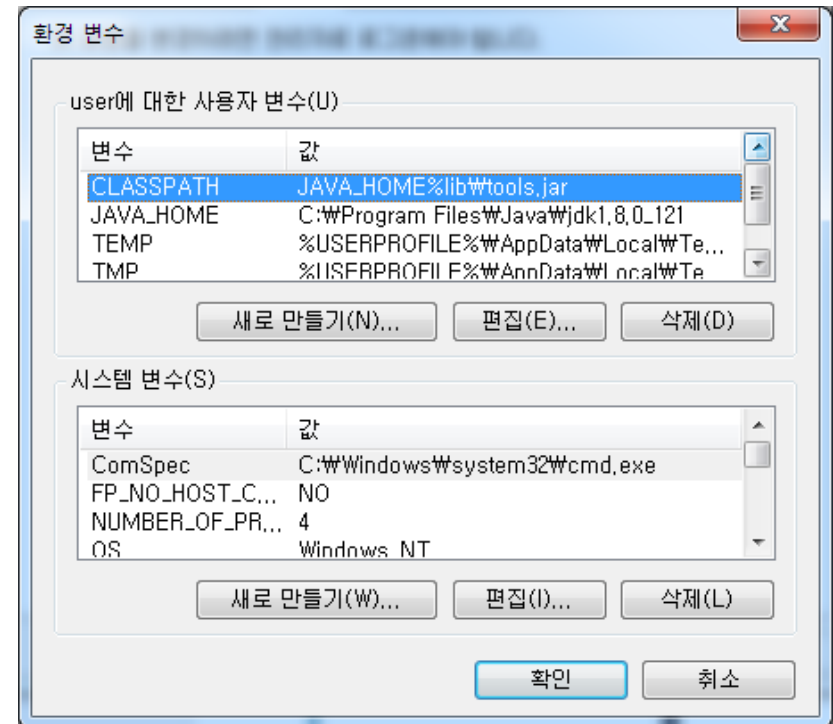
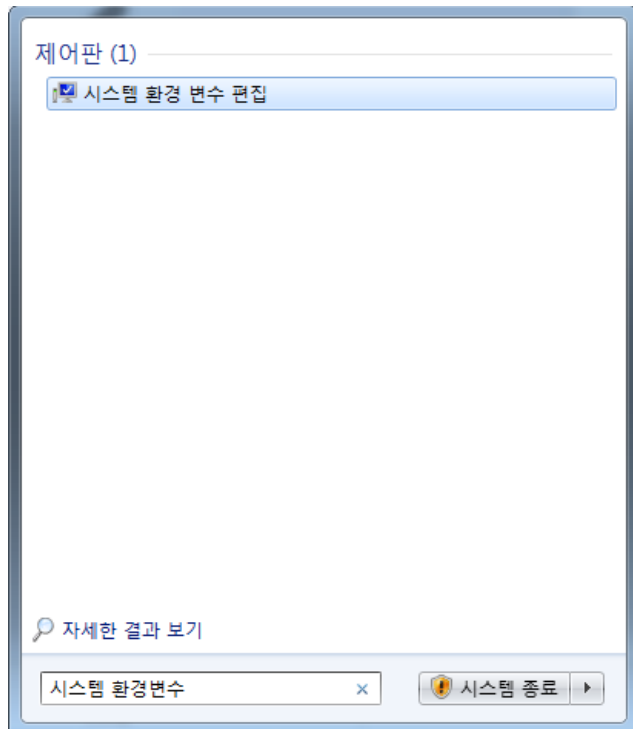
- Notice that a package name is not arbitrary, but must be **a list of directories** leading from **a class path base** directory **to the package classes**. Thus, the package name tells Java what subdirectories to go through—starting from the base class directory—to find the package classes.

- Java searches from the path in the ‘class path variable’ for example,

**CLASSPATH=/myjavastuff/libraries:/home/username/**

# A Package Name

Remember we set the path through control panel in Windows OS.

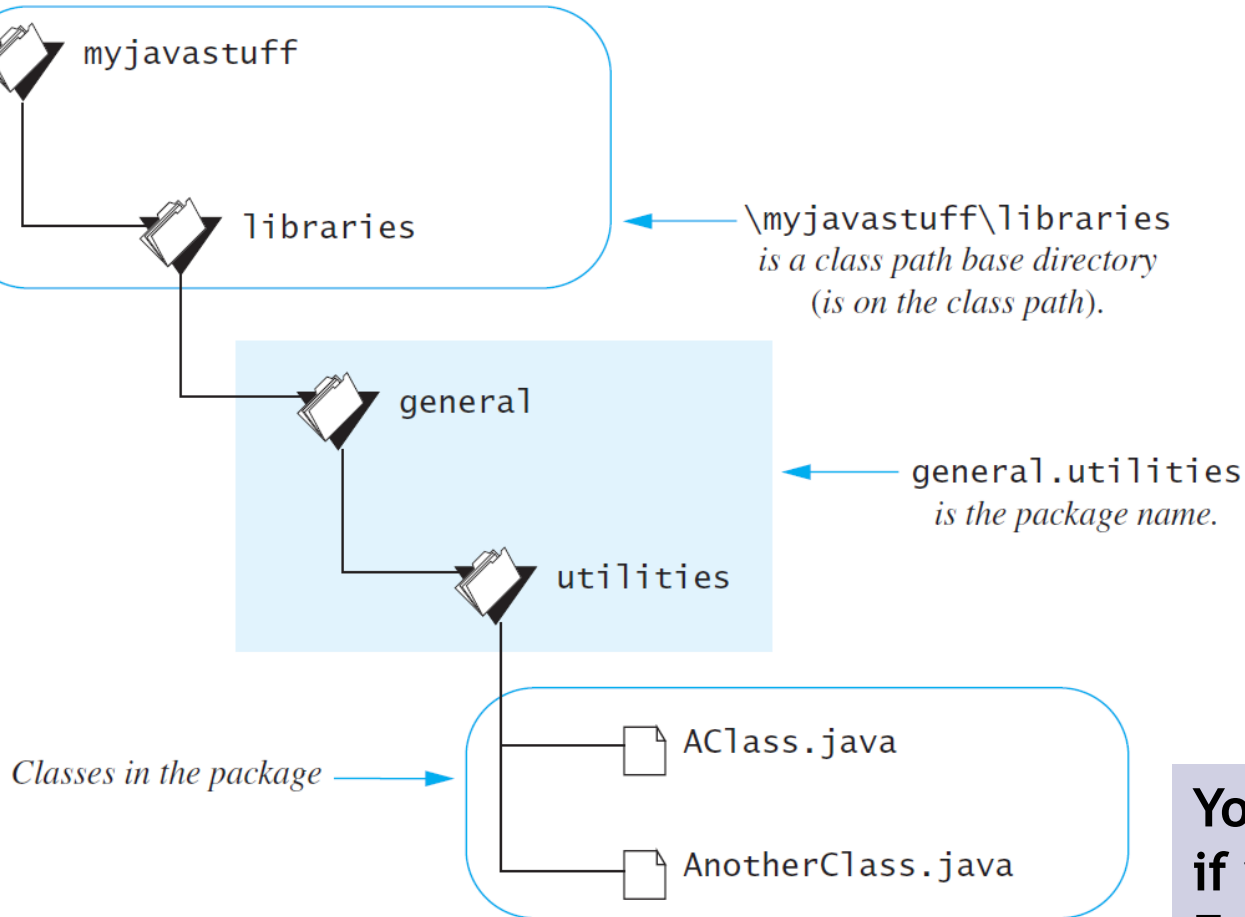


Add new Environment variables

Variable	Value
JAVA_HOME	C:\Program Files\Java\jdk1.8.0_121
CLASSPATH	JAVA_HOME%lib%tools.jar



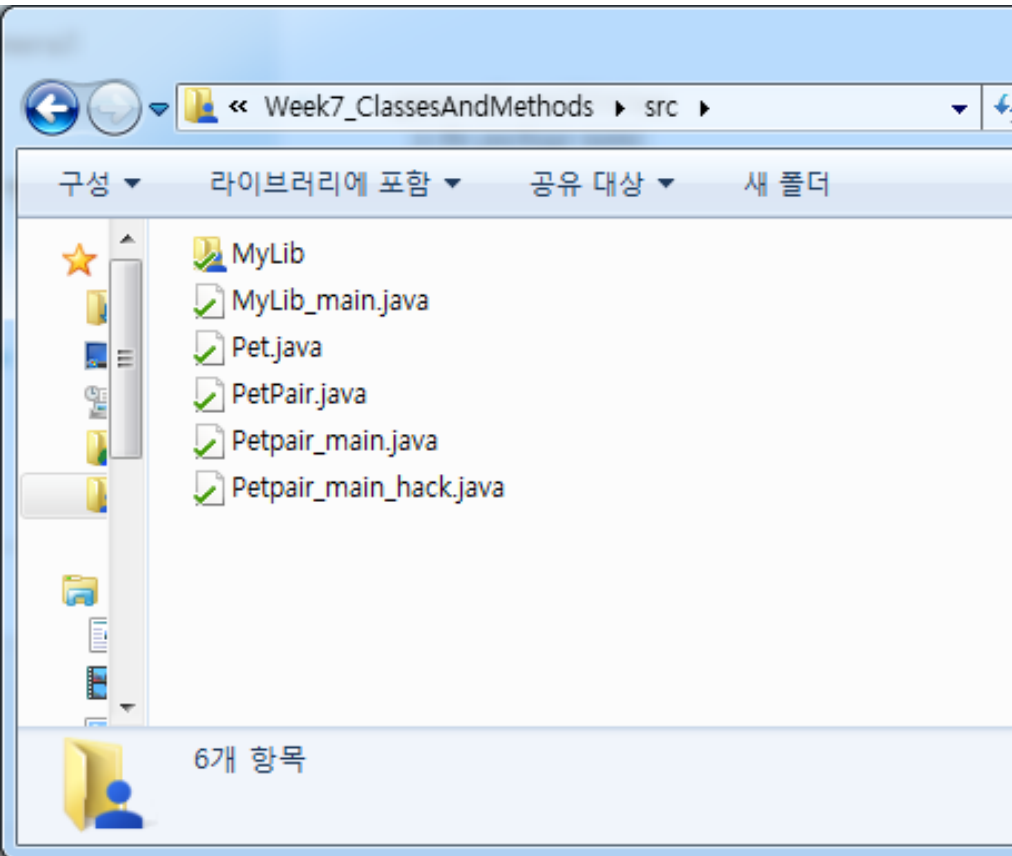
# A Package Name



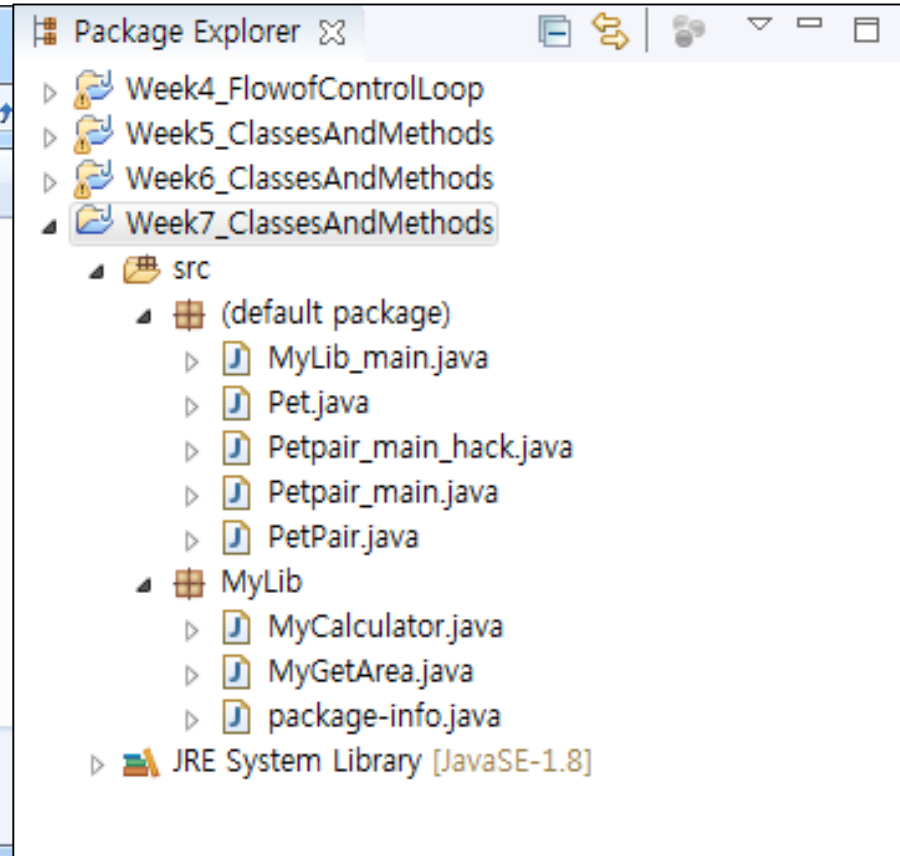
You can use the classes if you import the package.  
Ex) ***general.utilities***

# A Package Name

## In Window Explorer



## In Eclipse



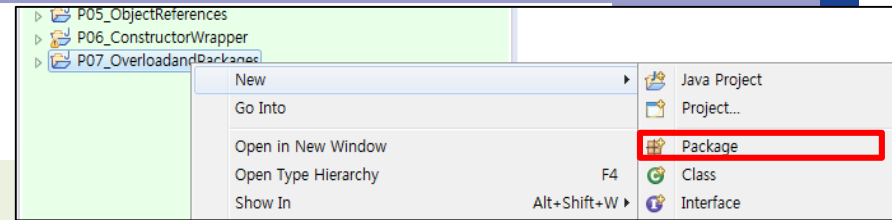
# Name Clashes

---



- Packages help in dealing with name clashes
  - When two classes have same name
- Different programmers may give same name to two classes
  - Ambiguity resolved by using the package name

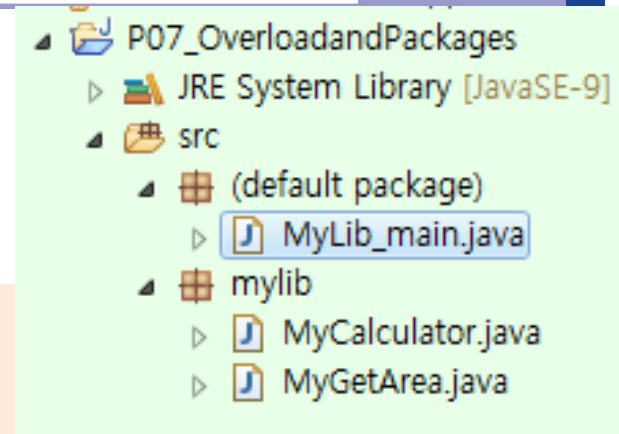
# Create “mylib” and write two classes



```
package mylib;
public class MyGetArea {
    public static double Square(double width, double height){
        return width*height;
    }
    public static double Triangle(double width, double height){
        return width*height/2;
    }
    public static double Circle(double radius){
        return radius*radius*3.14;
    }
}
```

```
package mylib;
public class MyCalculator {
    public static double addition(double a, double b)           {return a+b;}
    public static double subtraction(double a, double b)        {return a-b;}
    public static double division(double a, double b)            {return a/b;}
    public static double multiplication(double a, double b)      {return a*b;}
}
```

# Write a main class “MyLib\_main.java” in the default package



```
import mylib.*;
public class MyLib_main {
    public static void main(String[] args) {
        double width      = 12;
        double height      = 11;

        double square_area    = MyGetArea.Square(width, height);
        double triangle_area   = MyGetArea.Triangle(width, height);
        double sum = MyCalculator.addition(width, height);
        System.out.println("Square: " + square_area);
        System.out.println("Triangle: " + triangle_area);
        System.out.println("sum: " + sum);
    }
}
```