

# Object – Oriented Programming

## Lab #3

# ● Contents

- **Classes**
  - class
  - object
  - method
- The *new* operator
- The *this* operator
- Method Overloading
- Constructor

CSLAB

# ● Classes and Objects

- **Class:** A class is a blueprint or prototype from which objects are created
- **Object:** An object is an instance of a class. The relationship is such that many objects can be created using one class. Each object has its own data but its underlying structure (i.e., the type of data it stores, its behaviors) are defined by the class. An object has both data and actions (methods).



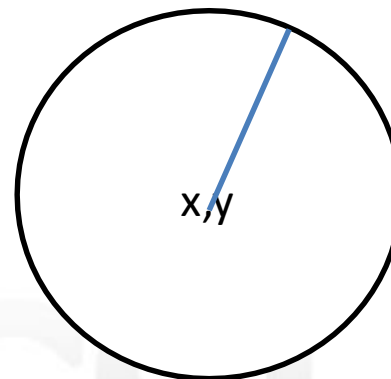
CSLAB

# ● Classes

- A *class* is a collection of *fields* (data) and *methods* (procedure or function) that operate on that data

Circle
<i>centre</i> <i>radius</i>
<i>circumference()</i> <i>area()</i>

Instance of the  
circle class



CSLAB

## ● Classes (Contd)

- The basic syntax for a class definition:

```
class ClassName [extends SuperClassName]  
{  
    [fields declarations]  
    [methods declarations]  
}
```

- Bare bone class – no fields, no methods

```
public class Circle {  
    // my circle class  
}
```

CSLAB

## ● Adding Fields: Class Circle with fields

- Add *fields*

```
public class Circle {  
    public double x, y; //centre coordinate  
    public double r; // radius of the circle  
}
```

Instance Variables

Instance Variables Type

Modifiers: Specifies the visibility of the instance variable.

- The fields(data) are also called the *instance* variables

CSLAB

## ● Adding Methods

- A class with only data fields has no life. Objects created by such a class cannot respond to any messages.
- Methods are declared inside the body of the class but immediately after the declaration of data fields.

CSLAB

## ● Adding Methods (Contd)

- Method definition have the following parts

- Modifier
  - Return type
  - Method name
  - List of parameters
  - Method body
- } Method signature

- The general form of a method declaration is:

```
Modifier ReturnType MethodName(parameter-list)  
{  
    Method-body Statements;  
}
```

CSLAB



# ● Adding Methods to Class Circle

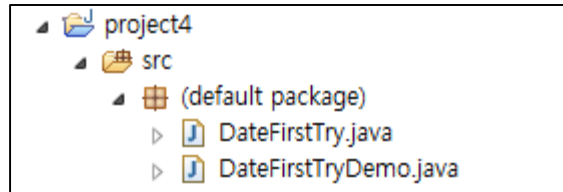
```
public class Circle {  
  
    public double x, y; // centre of the circle  
    public double r; // radius of circle  
  
    //Methods to return circumference and area  
    public double circumference() {  
        return 2 * 3.14 * r;  
    }  
    public double area() {  
        return 3.14 * r * r;  
    }  
}
```

Method Body



CSLAB

# Classes - Example



```
public class DateFirstTry
{
    public String month;
    public int day;
    public int year; //a four digit number.

    public void writeOutput( )
    {
        System.out.println(month + " " + day + ", " + year);
    }
}
```

```
public class DateFirstTryDemo
{
    public static void main(String[] args)
    {
        DateFirstTry date1, date2;
        date1 = new DateFirstTry( );
        date2 = new DateFirstTry( );
        date1.month = "December";
        date1.day = 31;
        date1.year = 2007;
        System.out.println("date1:");
        date1.writeOutput( );

        date2.month = "July";
        date2.day = 4;
        date2.year = 1776;
        System.out.println("date2:");
        date2.writeOutput( );
    }
}
```

- Note: You need to create 2 classes in the project, one with name DateFirstTry and one with name DateFirstTryDemo which is the main class

CSLAB

# • The *new* Operator

- The *new* Operator

- The *new* operator is used to create an *object* of a class and associate the object with a variable that names it

- Syntax

- *Class variable = new Class();*

- Example

- *DateFirstTry date1 = new DateFirstTry();*

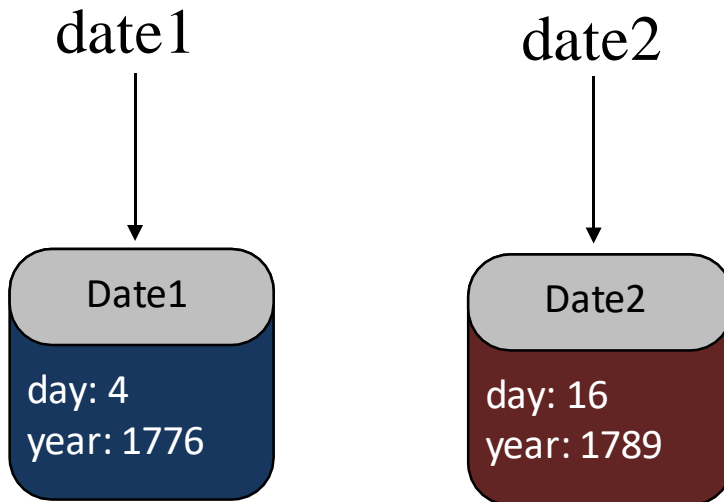
CSLAB

- Creating objects of a class

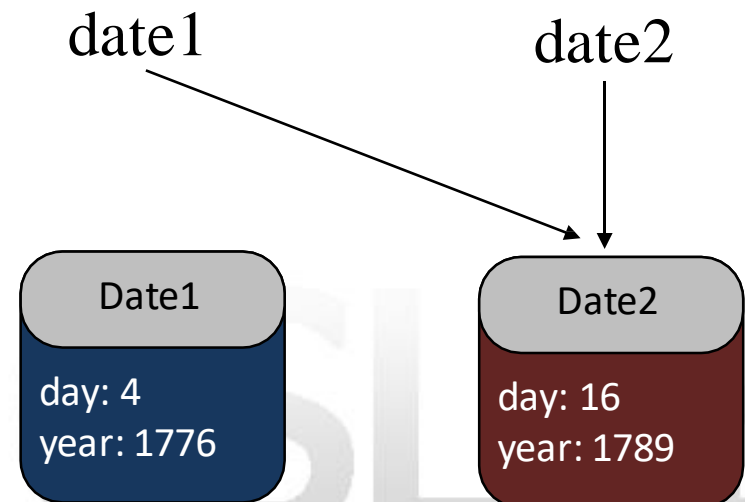
```
Date date1 = new Date(4, 1776);  
Date date2 = new Date(16, 1789);
```

```
date1 = date2;
```

Before Assignment



After Assignment



# ● Instance Variable and Methods

- In order to use an object we can reference its instance variables or methods using dot(.) notation
- *objectIdentifier.variableIdentifier*
- *objectIdentifier.methodName*
- We can use this to
  - get information from the variables
  - Set variables to specific values
  - Get the methods to perform actions
- This, however, depends on the *modifier*

CSLAB

## • The *return* keyword

- The *return* keyword immediately stops execution of a method
  - Jumps back to whatever called that method
  - Returns a value

- *void* methods do not return a value

- Consider the following method

```
public static void foo(int x) {  
    if (x==1) {  
        return;  
    }  
    System.out.println("x is not 1");  
}
```

- This method will only print the string if x is not 1
- How can this be modified to return a string?

CSLAB

# ● Visibilities in Java

- There are four visibilities:

- ***private***: Only code within the same class can access the field or method
  - Note: "access" means reading or writing the field, or invoking the method
- ***public***: Any code, anywhere, can access the field or method
- ***protected***: Used with inheritance
  - We will see it later
- ***default***: accessible in classes in the same package and the class itself

CSLAB

## ● A few notes on visibilities

- You can NOT specify visibilities for method variables
  - Any method variable can only be accessed within that method
- But, you can specify visibilities for methods and classes

CSLAB



# ● Variables

- There are several kinds of variables:
  - Member variables in a class – these are called *fields* or *instance variables*
  - Variables in a method or block of code – these are called *local variables*
  - Variables in method declarations – these are called *parameters*

CSLAB

## ● Self-Test (1)

- **GitHub에 있는 DateFirstTry에 다음과 같은 메소드를 추가할 것**
  - makeltNewYears
    - instance variable month를 "January"로, day를 1로 바꾸는 메소드 (year는 바뀌지 않는다)
    - 매개변수를 가지지 않음
    - return type은 void
  - yellIfNewYear
    - instance variable month가 "January"이고 day가 1일 경우 "Happy New Year!"를, 그렇지 않을 경우 "Not New Year's Day."를 출력하는 메소드
    - 매개변수를 가지지 않음
    - return type은 String
- **DateFirstTryDemo에서 DateFirstTry 클래스의 메소드를 수행하여 다음과 같은 내용을 출력할 것**
  - date1: Not New Year's Day.
  - date2: Happy New Year!!!

CSLAB

## ● The *this* Reference

- In Java, the *this* reference is used to refer to the object you are *inside* of at this moment
- We say that each object has a reference to itself
  - called the *this* reference

CSLAB

## • The *this* Reference (Contd)

```
public class test {  
  
    String Name;  
    int age;  
  
    public void set(String Name,int age)  
    {  
        this.Name = Name;  
        this.age = age;  
    }  
  
    public void show()  
    {  
        System.out.println(this.Name);  
        System.out.println(this.age);  
    }  
  
    public static void main(String[] args)  
    {  
        test t1 = new test();  
        t1.set("alice", 23);  
        t1.show();  
    }  
}
```

I am **this**  
object.  
My name is  
alice and I am  
23.

test1

name: Fred  
age: 25

CSLAB

# ● Concepts

- **Information Hiding** is the practice of separating how to use a class from the details of its implementation
  - Abstraction is another term used to express the concept of discarding detail in order to avoid information overload
- **Encapsulation** means that the data and methods of a class are combined into a single unit (i.e., a class object), which hides the implementation details
  - Knowing the details is unnecessary because interaction with the object occurs via a well-defined and simple **interface**
  - In Java, hiding details is done by marking them **private**

CSLAB

## ● Accessor-Mutator methods

- An **accessor method (getter)** is used to return the value of a private field
- It follows a naming scheme prefixing the word “**get**” to the start of the method name
- A **mutator method (setter)** is used to set a value of a private field
- It follows a naming scheme prefixing the word “**set**” to the start of the method name

CSLAB

## ● Classes – Example2 (Contd)

```
public class DateSecondTry
{
    private String month;
    private int day;
    private int year;
    .
    .
    .

    public void setDate(String month, int day, int year)
    {
        .
        .
        .
        this.month = month;
        this.day = day;
        this.year = year;
    }

    public void setDate(String month, int day)
    {
        .
        .
        .
        this.month = month;
        this.day = day;
        this.year = 2000;
    }

    public void setDate(int year)
    {
        .
        .
        .
        setDate(1, 1, year);
    }
}
```

```
public class DemoOfDateSecondTry
{
    public static void main(String[] args)
    {
        DateSecondTry date1 = new DateSecondTry( );
        date1.setDate("November",9,1990);
        date1.writeOutput();
        date1.setDate("April",6);
        date1.writeOutput();
        date1.setDate(2012);
    }
}
```

CSLAB

# ● Method Overloading

- Methods must all have different headers
  - Methods within a class can have the same name if they have different parameter lists
  - Differentiated by the number and the type of the arguments passed into the method
  - We've seen overloading before:
    - $3 + 4$                       Performs integer addition
    - $3.0 + 4.0$                 Performs floating-point addition
    - $"3" + "4"$                 Performs string concatenation
- => The '+' operator is overloaded

CSLAB



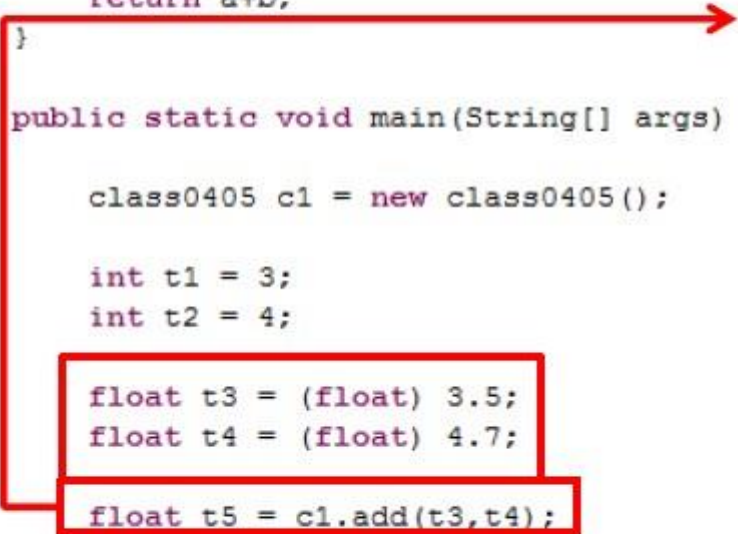
# ● Overloading Example

```
public class class0405{  
    public void add(int a, int b)  
    {  
        System.out.println("더한 결과는 [" + (a+b) + "]입니다");  
    }  
    public float add(float a, float b)  
    {  
        return a+b;  
    }  
    public static void main(String[] args) {  
        class0405 c1 = new class0405();  
        int t1 = 3;  
        int t2 = 4;  
        float t3 = (float) 3.5;  
        float t4 = (float) 4.7;  
        float t5 = c1.add(t3,t4);  
        c1.add(t1,t2);  
        System.out.println("더한 결과는 [" + t5 + "]입니다");  
    }  
}
```

AB

# ● Overloading Example (Contd)

```
public class class0405{  
  
    public void add(int a, int b)  
    {  
        System.out.println("더한 결과는 [" + (a+b) + "]입니다");  
    }  
  
    public float add(float a, float b)  
    {  
        return a+b;  
    }  
  
    public static void main(String[] args) {  
  
        class0405 c1 = new class0405();  
  
        int t1 = 3;  
        int t2 = 4;  
  
        float t3 = (float) 3.5;  
        float t4 = (float) 4.7;  
  
        float t5 = c1.add(t3,t4);  
        c1.add(t1,t2);  
        System.out.println("더한 결과는 [" + t5 + "]입니다");  
    }  
}
```



AB

# • Constructor

- A constructor is a special member function whose name is always the same as the name of the class
- A constructor is invoked each time a *new* object is created
- Usually used for initializing variables
- The constructor **cannot** have any return type – not even void
- The “default” constructor accepts no arguments
- The constructor is usually overloaded

CSLAB

# Classes - Example3

```
public class Date
{
    private String month;
    private int day;
    private int year; //a four digit number.

    public Date( )
    {
        month = "January";
        day = 1;
        year = 1000;
    }

    public Date(String month, int day, int year)
    {
        setDate(month, day, year);
    }

    public Date(int month, int day, int year)
    {
        String strMonth = Integer.toString(month);
        // converts int to string
        setDate(strMonth, day, year);
    }

    public Date(int year)
    {
        setDate(1, 1, year);
    }
}
```

```
public class ConstructorsDemo
{
    public static void main(String[] args)
    {
        Date date1 = new Date("December", 16, 1770),
            date2 = new Date(1, 27, 1756),
            date3 = new Date(1882),
            date4 = new Date( );
    }
}
```

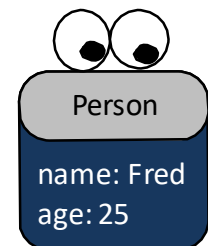
Overloaded constructors

CSLAB

## • equals()

- It is expected by java that objects be able to determine if they are equal to another.
  - The equals() method returns a Boolean
  - True if the object is the same. False if it is not.
  - The programmer must determine the criteria for objects being equal.

```
public boolean equals(Person anotherPerson){  
    if (this.name.equals(anotherPerson.name) && this.age == anotherPerson.age)  
        return true;  
    else  
        return false;  
}
```



CSLAB

# • toString()

- It is useful for an object to return a string representation of itself.
  - The toString() method returns this String.
  - The programmer must determine the information that is returned by the method.

```
public String toString() {  
    return "Person:: \n" + "Name: "+this.name+"\n" + "Age: "+this.age+"\n";  
}
```



CSLAB

## ● Method notes

- You can put the methods in a class in any order
  - Java doesn't care which one is listed first
  - Thus, you can call a method listed later within another method
- All methods must specify a return type
  - If it's void, then no value is returned
- Parameters can't be changed within a method

CSLAB

## ● Class Outline

```
public class ClassName{  
    instance variables;  
    Constructors(){...}  
    getters(){...} //accessors  
    setters(parameters){...} //mutators  
    other methods(){...}  
    equals(){...}  
    toString(){...}  
}
```

CSLAB



# ● Class Outline (Contd)

Display 5.11 A Simple Class

```
1 public class ToyClass
2 {
3     private String name;
4     private int number;
5
6     public ToyClass(String initialName, int initialNumber)
7     {
8         name = initialName;
9         number = initialNumber;
10
11     public ToyClass()
12     {
13         name = "No name yet.";
14         number = 0;
15
16     public void set(String newName, int newNumber)
17     {
18         name = newName;
19         number = newNumber;
20
21     public String toString()
22     {
23         return (name + " " + number);
24
25     public static void changer(ToyClass aParameter)
26     {
27         aParameter.name = "Hot Shot";
28         aParameter.number = 42;
29
30     public boolean equals(ToyClass otherObject)
31     {
32         return ((name.equals(otherObject.name))
33             && (number == otherObject.number));
34 }
```

instance variables

constructors

setter (mutator)

toString()

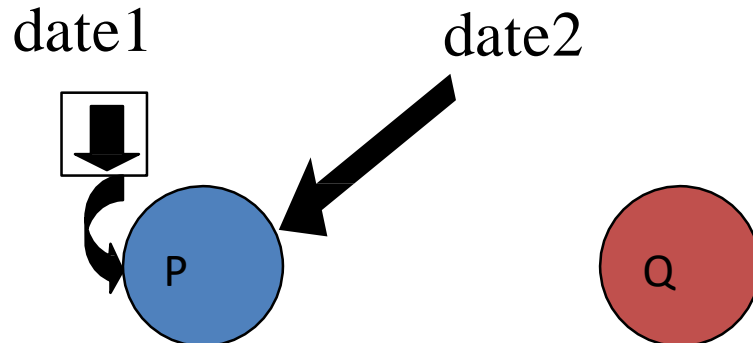
other methods

equals()

CSLAB

# Automatic garbage collection

- The object which does not have a reference and cannot be used in future
- The object becomes a candidate for automatic **garbage collection**
- Java automatically collects garbage periodically and releases the memory used to be used in future



The Q has no reference variable  
so it will be collected by the  
**Garbage Collector**

## ● Self-Test (2)

### • Employee 클래스를 생성할 것

- employee의 String name, int age, String position, int salary, int vacationDays 를 저장하기 위한 private variable을 만들 것
- 3개의 생성자를 만들 것
  - default 생성자
  - name, age만 설정하는 생성자
  - 모든 instance variable을 설정하는 생성자
  - 만약 position이 설정되지 않을 경우 default는 "Engineer"로 설정됨
  - 만약 salary가 설정되지 않을 경우 default는 15000로 설정됨
  - 만약 vacationDays가 설정되지 않을 경우 default는 20으로 설정됨
- instance variable salary를 변경하는 mutator method를 만들 것
- employee의 모든 instance variable을 출력하는 public method outInfo() 를 만들 것
- int type 값을 매개변수로 받는 public method vacation() 를 만들 것
  - employee가 휴가를 신청할 때 사용하는 메소드
  - 인자보다 vacationDays가 크거나 같을 경우 vacationDays가 인자만큼 차감되고 "Possible"을 return함
  - 인자보다 vacationDays가 부족할 경우 vacationDays는 차감되지 않고 "Impossible"을 return함

CSLAB

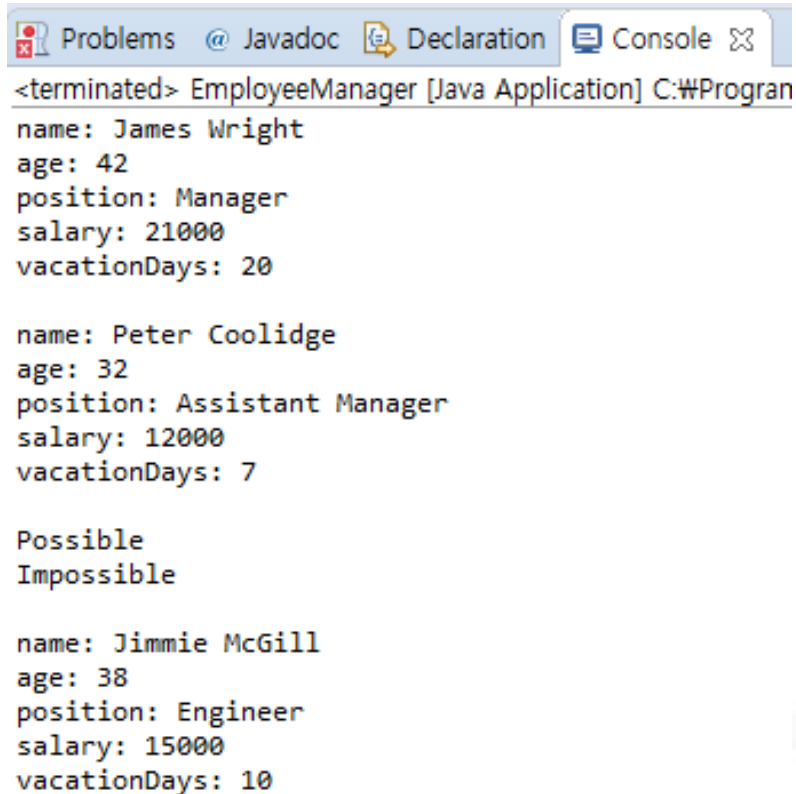
## ● Self-Test (2) (Contd)

### • EmployeeManager를 생성할 것

- Employee 클래스의 메소드를 통해 다음과 같이 객체의 정보를 수정하고 출력할 것
  - James Wright의 Salary를 21000으로 변경할 것
  - James Wright과 Peter Coolidge의 정보를 출력할 것
  - Jimmie가 휴가를 10일 신청할 경우를 출력할 것
  - Peter가 휴가를 10일 신청할 경우를 출력할 것
  - Jimmie의 정보를 출력할 것

CSLAB

## ● Self-Test (2) (Contd)



```
<terminated> EmployeeManager [Java Application] C:\#Progran
name: James Wright
age: 42
position: Manager
salary: 21000
vacationDays: 20

name: Peter Coolidge
age: 32
position: Assistant Manager
salary: 12000
vacationDays: 7

Possible
Impossible

name: Jimmie McGill
age: 38
position: Engineer
salary: 15000
vacationDays: 10
```

CSLAB