Lab 3
OOP – Abstraction, Polymorphism, Inheritance, Encapsulation
Constructor

- Form of the Constructor
  - The class has a function that is named same as class name.
  - Return type not declared, not actually returned.
  - A kind of function that allows initializing variables (using parameters)
Constructor

생성자

- new 연산자에 의해 호출되는 중괄호 블록
- 객체 생성시 호출됨
- 객체 초기화 담당

기본 생성자

- 선언 생략시 컴파일러가 자동으로 바이트 코드에 추가

명시적 생성자 선언

- 객체 다양한 값으로 초기화 목적

생성자 오버로딩

- 매개 변수를 달리하는 생성자 여러개 선언하는 것

```java
public class Car{
    String model, color;
    int max;

    Car(String model){
        this(model, "silver", 250);
    }
    Car(String model, String color){
        this(model, color, 250);
    }
    Car(String model, String color, int max){
        this.model= model;
        this.color= color;
        this.max= max;
    }
}
```
Constructor

```java
public class ConstructorEx5 {
    public ConstructorEx5() {
        System.out.println("default constructor");
    }

    public ConstructorEx5(String str) {
        this(str, 10);
        System.out.println(str + " constructor");
    }

    public ConstructorEx5(String str, int n) {
        this();
        System.out.println(str + " + n + constructor");
    }

    public static void main(String[] args) {
        new ConstructorEx5("new constructor");
    }
```
## Memory Access Control

<table>
<thead>
<tr>
<th>접근제한자</th>
<th>접근 불가한 클래스</th>
</tr>
</thead>
<tbody>
<tr>
<td>public</td>
<td>x</td>
</tr>
<tr>
<td>protected</td>
<td>자식 클래스가 아닌 다른 패키지에 소속된 클래스</td>
</tr>
<tr>
<td>default</td>
<td>다른 패키지에 소속된 클래스</td>
</tr>
<tr>
<td>private</td>
<td>all</td>
</tr>
</tbody>
</table>
1. Abstraction

- Abstraction is a process where you show only “relevant” data and “hide” unnecessary details of an object from the user.
Abstraction - example

```java
public class Sketch{
    public static void main(String []args) {
        // Shape a = new Shape(); compile error
        Shape a = new Rectangle();
        a.onDraw();
        a.onDelete();

        Shape b = new Circle();
        b.onDraw();
        b.onDelete();
    }
}

abstract class Shape{
    abstract public void onDraw();
    abstract public void onDelete();
}

class Rectangle extends Shape{
    public void onDraw() { System.out.println("draw rect"); }
    public void onDelete() { System.out.println("delete rect"); }
}

class Circle extends Shape{
    public void onDraw() { System.out.println("draw cir"); }
    public void onDelete() { System.out.println("delete cir"); }
}
```
2. Encapsulation

• Encapsulation is a process where you keep all the inner works of the system together hidden. The important works are stored hidden to keep it safe from the average user which ensures the integrity of the system as it was designed.
2. Encapsulation

```java
private 서터 닫힘() {
    매우 복잡한 코드;
}

private 메모리 저장() {
    매우 복잡한 코드;
}

public 사진 찍는 버튼() {
    ~
    서터 닫힘();
    메모리 저장();
    ~
}
```
Encapsulation

Need to prevent outside code from accessing data that is private.

Therefore, you need getter and setter for this purpose, when you program!

Setter: method that takes parameter(s) and assign them to private variable(s) (It prevents outside from directly modifying the data)

Getter: method that takes no parameter and just returns values of private variable(s) (It prevents outside from directly retrieving the data)
Encapsulation - example

```java
package cp2021;

public class Book {
    private String title;
    private String author;

    public Book() { super(); }
    public Book(String title, String author) {
        super();
        this.title = title;
        this.author = author;
    }

    public String getTitle() { return title; }
    public void setTitle(String title) { this.title = title; }
    public String getAuthor() { return author; }
    public void setAuthor(String author) { this.author = author; }

    public String toString() { return "title is " + title + "$author is " + author; }
}
```
package cp2021;

public class Library {
    public static void main(String[] args) {
        System.out.println("package 2021");
        // book1.title="java" compile error
        book1.setTitle("java");
        book1.setAuthor("See");
        System.out.println(book1.toString());

        System.out.println(book2.getTitle());
        System.out.println(book2.getAuthor());
    }
}

3. Inheritance

• Inheritance is the mechanism by which an object acquires the some/all properties of another object.

• It supports the concept of hierarchical classification.
3. Inheritance
Inheritance - example

```java
class ParentEx{
    public ParentEx() {
        super();
        System.out.println("parent class");
    }
}

class ChildEx extends ParentEx{
    public ChildEx() {
        super();
        System.out.println("child class");
    }
}

public class InheritanceEx2 {
    public static void main(String[] args) {
        new ChildEx();
    }
}
```
4. Polymorphism

- Polymorphism means to process objects differently based on their data type.
- One method with multiple implementation, for a certain class of action.
4. Polymorphism
Polymorphism - example

```java
package cp2021.Animal;

public class Animal {
    public void speak() { System.out.println("can't speak"); }
    public void walk() { System.out.println("4 legs"); }
}
```
Polymorphism - example

```java
package cp2021.Animal;

public class Cat extends Animal {
    public void speak() { System.out.println("meow"); }
}

package cp2021.Animal;

public class Dog extends Animal {
    public void speak() { System.out.println("woof"); }
}

package cp2021.Animal;

public class Duck extends Animal {
    public void speak() { System.out.println("quack"); }
    public void walk() { System.out.println("2 legs"); }
}

package cp2021.Animal;

public class Fish extends Animal {
    public void walk() { System.out.println("can't walk"); }
}
```
package cp2021.Animal;

import java.util.Scanner;

// static binding : 컴파일 시점에 호출 메서드를 알고 있는 경우
public class AnimalMain1 {

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        int n;
        Dog ob1 = null;
        Cat ob2 = null;
        Duck ob3 = null;
        Fish ob4 = null;

        while (true) {
            System.out.print("1.Dog 2.Cat 3.Duck 4.Fish 5.Exit\ynSelect:");
            n = sc.nextInt();

            switch (n) {
                case 1:
                    ob1 = new Dog();
                    ob1.speak();
                    ob1.walk();
                    break;
                case 2:
                    ob2 = new Cat();
                    ob2.speak();
                    ob2.walk();
                    break;
                case 3:
                    ob3 = new Duck();
                    ob3.speak();
                    ob3.walk();
                    break;
                case 4:
                    ob4 = new Fish();
                    ob4.speak();
                    ob4.walk();
                    break;
                default:
                    System.out.println("종료합니다.");
                    sc.close();
                    System.exit(0);
                    break;
            }
        }
    }
}

DCSLAB CSE, SNU
package cp2021.Animal;

import java.util.Scanner;

// 동적 바인딩 (Dynamic Binding): 실행시점 (run time)에 호출 메서드를 결정한다.
// 코드량을 상당부분 줄일 수 있다.

public class AnimalMain2 {
    public static void main(String [] args) {
        Scanner sc = new Scanner(System.in);
        int n;
        Animal ani = null; // 부모 클래스로 만든 참조 변수

        while (true) {
Select:");
            n = sc.nextInt();

            switch (n) {
                case 1:
                    ani = new Dog();
                    break;
                case 2:
                    ani = new Cat();
                    break;
                case 3:
                    ani = new Duck();
                    break;
                case 4:
                    ani = new Fish();
                    break;
                default:
                    System.out.println("종료합니다.");
                    sc.close();
                    System.exit(0);
                    break;
            }
            ani.speak();
            ani.walk();
        }
    }
}
Exercise
Let’s write classes to practice inheritance & polymorphism

1. **Class GameCharacter.**
   a. It will serve as a parent class
   b. It has five variables: strength, dexterity, intelligence, luck, power (double type)
   c. Getter and setters for all variables (I know it is plenty.. But do it for your own benefit)
   d. It need constructors
      i. Default constructor that takes no parameter and initializes variables to 0
      ii. Overridden constructor that takes five parameters to initialize all variables.
   e. Virtual void attack() method
      i. It prints out “attack!!”
   f. Virtual double damage() method
      i. Leave it empty for now
Exercise

2. Class Warrior

a. It inherits from GameCharacter class.
b. Constructor that takes two parameters (default param1 value=10 and param2 value=30)
   i. Assign 1st parameter value to strong (larger than 10)
   ii. Assign 2nd parameter value to power (larger than 30)
   iii. Assign the rest to value of (parameter1 - 10)
c. void attack() method
   i. It prints out “Warrior: attack!! Clang! Clang!”
   ii. Also prints out damage
d. double damage() method
   i. It calculates damage based on variables and returns it.
   ii. Equation: damage = (strong*4 + dexterity + luck*0.1 + intelligence *0.1) * power
Exercise

3. Class Magician

a. It inherits from GameCharacter class.
b. Constructor that takes two parameters (default param1 value=10 and param2 value=30)
   i. Assign 1st parameter value to intelligence (larger than 10)
   ii. Assign 2nd parameter value to power (larger than 30)
   iii. Assign the rest to value of (parameter1 - 10)
c. void attack() method
   i. It prints out “Magician: attack!! magic balt!”
   ii. Also prints out damage
d. double damage() method
   i. It calculates damage based on variables and returns it.
   ii. Equation: damage = (intelligence*4 + luck + dexterity*0.1 + strength*0.1) * power
Exercise

4. Class Bowman

a. It inherits from GameCharacter class.

b. Constructor that takes two parameters (default param1 value=10 and param2 value=30)
   i. Assign 1st parameter value to dexterity (larger than 10)
   ii. Assign 2nd parameter value to power (larger than 30)
   iii. Assign the rest to value of (parameter1 - 10)

c. void attack() method
   i. It prints out “Bowman: attack!! zing zing!”
   ii. Also prints out damage

d. double damage() method
   i. It calculates damage based on variables and returns it.
   ii. Equation: damage = (dexterity*4 + strength + intelligence*0.1 + luck*0.1) * power
Exercise

5. Class Thief

a. It inherits from GameCharacter class.

b. Constructor that takes two parameters (default param1 value=10 and param2 value=30)
   i. Assign 1st parameter value to luck (larger than 10)
   ii. Assign 2nd parameter value to power (larger than 30)
   iii. Assign the rest to value of (parameter1 - 10)

c. void attack() method
   i. It prints out “Thief: attack!! ping ping!”
   ii. Also prints out damage

d. double damage() method
   i. It calculates damage based on variables and returns it.
   ii. Equation: damage = (luck*4 + dexterity + intelligence*0.1 + strength*0.1) * power