Classes - Basics

- Classes
- Information hiding
- Method overloading
- Important methods: toString(), equals(), hashCode()
- Parameter passing: pass by value
- Final fields
- Static fields and methods
- Objects Initialization
- Reflection

Object-oriented programming

- 추상화 (Abstraction): 추상화는 어떤 특정 정보를 표시해야 하고 어떤 정보를 숨겨야 하는지 식별하는 데 도움이 되는 기술 ("핵심이 뭐야?", "어느 레벨에서 설명 드릴까요?")
  - 절차- 세부 사항을 무시하면서 필수 정보를 추출
  - 실체(entity) – 실제에 대한 모델(model), 뷰(view), 표현(representation)
- 캡슐화 (Encapsulation): 숨겨야 할 내용을 숨기고 표시할 내용을 표시하는 방식으로 정보를 패키징하는 기술 (예 - 함수, 클래스, 모듈 등, "캡슐화된 모든 것이 숨겨져 있는가?")
  - 절차 – 물리적 논리적으로 둘러싸는 행위
  - 실체(entity) – 패키지 혹은 인클로저

  ★ 정보 은닉(information hiding): 기능을 수행하는데 필요한 내부 구현이나, 변경될 가능성이 있는 의사결정의 내부 사항을 숨김

Object-oriented programming

- 상속 (Inheritance): 파생 클래스(derived class, 상속 받는 클래스)가 기본 클래스(base class)의 모든 멤버 변수와 멤버 함수를 포함함
- 다형성 (Polymorphism): 서로 다른 유형의 엔티티에 대한 단일 인터페이스를 제공하거나, 단일 기호를 사용하여 여러 유형을 나타내는 것 (동일한 함수 이름이 다르게 동작할 수 있음 - 함수 오버로딩, 클래스 멤버 함수 오버라이딩 등)

상속	다형성
기본적으로 클래스에 적용됨	기본적으로 함수나 클래스의 멤버 함수에 적용됨
코드의 재사용성을 높이고 코드 길이를 줄임	객체는 컴파일 타임이나 런타임에 어떤 형태의 함수를 실행할지 결정할 수 있음
single, hierarchical inheritance	컴파일 타임 다형성 (오버로드)과 런타임 다형성 (오버라이드)이 될 수 있음

https://www.geeksforgeeks.org/difference-between-inheritance-and-polymorphism/

Role, Responsibility, and Collaboration

- **요청(request)**과 **응답(response)**을 통해 다른 **객체(object)**와 **협력(collaboration)**

- 객체간 **협력**하는 과정 속에서 특정한 **역할(role)**을 부여받음
- **역할**은 어떤 **협력**에 참여하는 특정한 객체가 **그** 안에서 차지하는 **책임(responsibility)**이나 임무

객체 지향(object-oriented)

- 상태(state)와 행동(behavior)을 함께 지닌 자율적인 객체
- 적절한 책임(responsibility)을 수행하는 역할(role) 간의 유연하고 견고한 협력(collaboration) 관계를 구축
- 훌륭한 객체지향(object-oriented) 설계자가 되기 위한 도전
  - 코드를 담는 클래스의 관점에서 **메시지를 주고 받는 객체의 관점으로 사고의 중심을 전환**하는 것
  - 어떤 클래스가 필요한가가 아니라 **어떤 객체들이 어떤 메시지를 주고 받으며 협력하는가에 집중**
- 객체지향은 객체를 지향하는 것이지 클래스를 지향하는 것이 아님

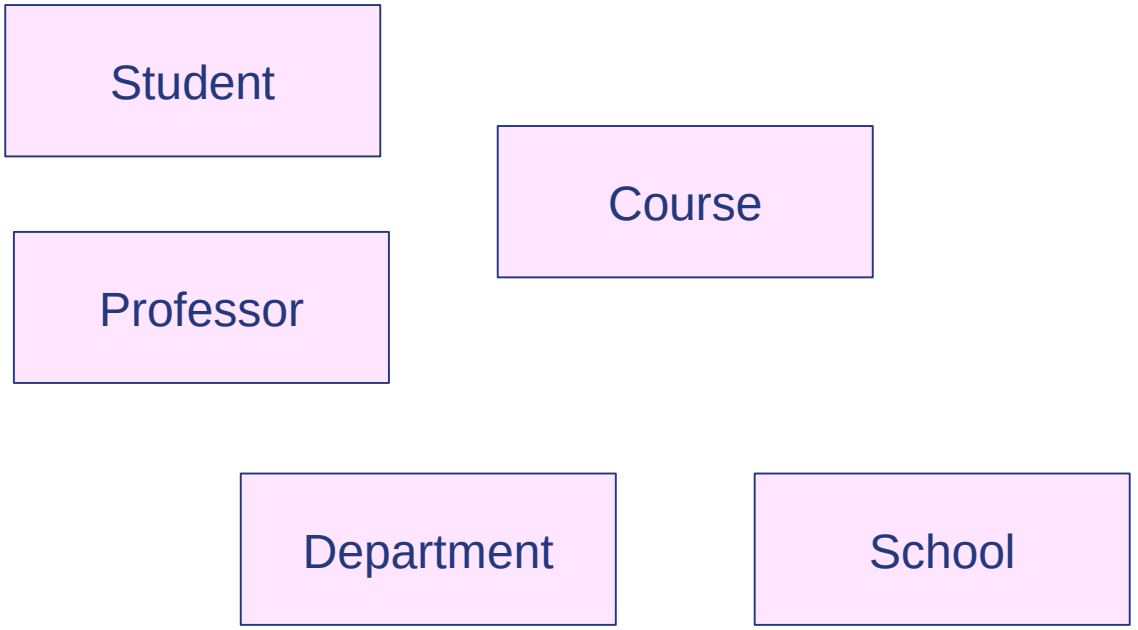
객체	클래스
타입(type)은 객체를 분류하는 기준 예) 프린터, 컴퓨터, 모니터	타입(type)을 구현할 수 있는 여러가지 구현 방법 중 하나 (자바 스크립트는 ES5까지 객체를 prototype 이용해서 구현)
객체가 어떤 타입에 속하는지를 결정하는 것은 객체가 수행하는 행동 예) 출력하는 행동을 하는 객체는 프린터 타입으로 분류	-
프로그램이 실행되어 객체의 상태 변경을 디버깅하는 시점 (동적인 모델)	프로그래밍 언어를 이용해 클래스를 작성하는 시점 (정적인 모델)

조영호, 객체지향의 사실과 오해, 위키북스, 2015

5

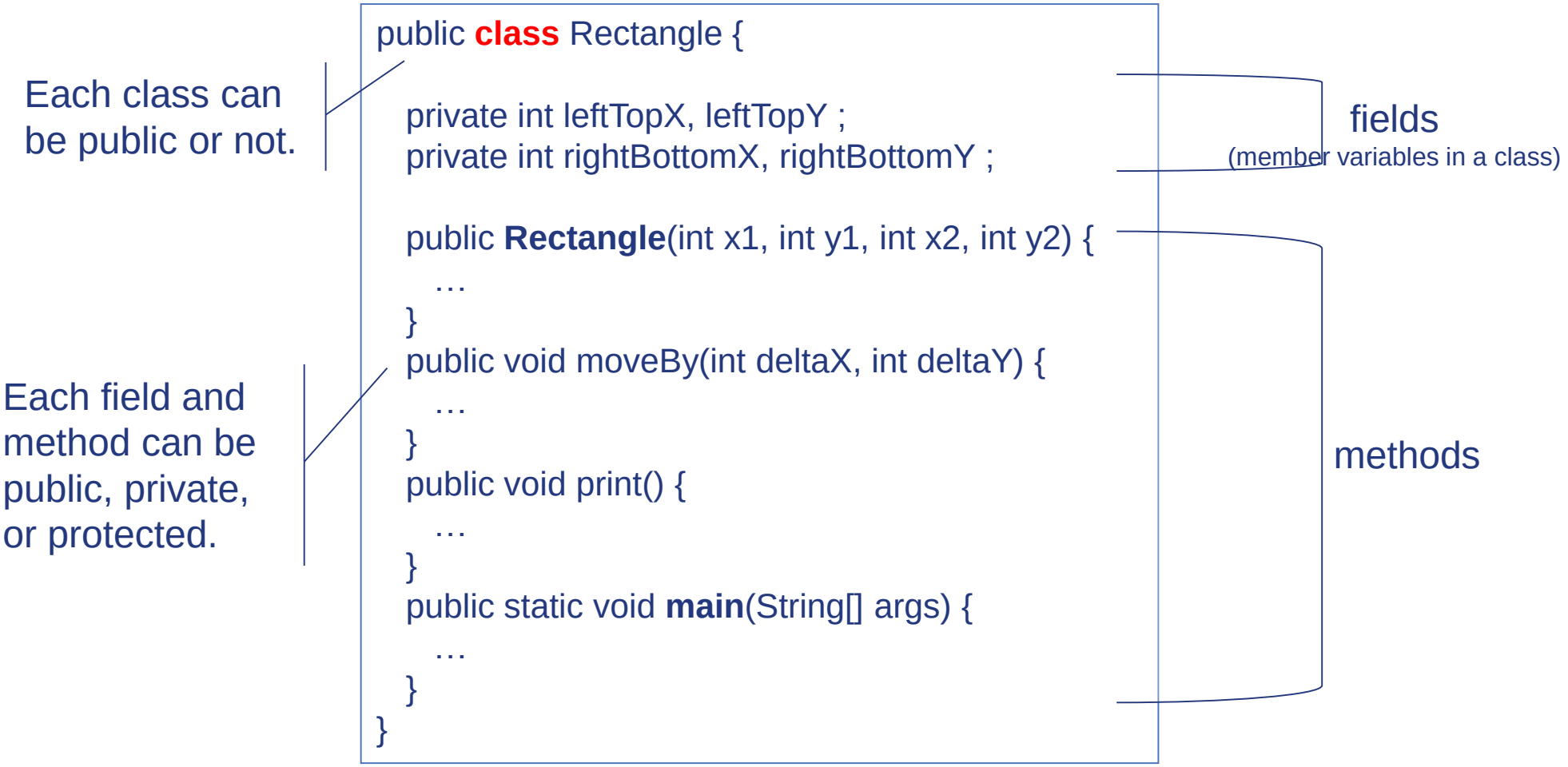
Class

- A class is an unit of Java programs; that is, Java programs consist only of classes.



Class

- Each class consists of fields and methods



Class: Rectangle

- Methods are implemented within the class.

```
public class Rectangle {
   private int leftTopX, leftTopY ;
   private int rightBottomX, rightBottomY ;

   public Rectangle(int x1, int y1, int x2, int y2) {
      leftTopX = x1 ; leftTopY = y1 ;
      rightBottomX = x2 ; rightBottomY = y2 ;
   }
   public void moveBy(int deltaX, int deltaY) {
      leftTopX += deltaX ; rightBottomY += deltaY ;
   }
   public void print() {
      System.out.printf("(%6d,%6d), (%6d,%6d)%n",
         leftTopX, leftTopY, rightBottomX, rightBottomY) ;
   }
   public static void main(String[] args) {
      Rectangle r = new Rectangle(10, 10, 200, 400) ;
      r.print();
      r.moveBy(50, 50) ;
      r.print();
   }
}
```

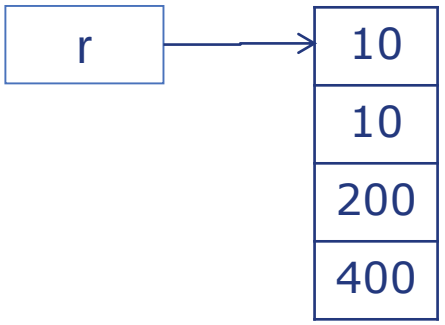
Constructor is used to initialize fields

Object should be created by new operator

Object Creation

- The process of creating an object from a class is called "instantiating" a class.
- In Java, objects can be created only through **new** operator.
  - Rectangle r(10, 10, 200, 400) is not allowed!

```
public class Rectangle {
   …
   public static void main(String[] args) {
      Rectangle r = new Rectangle(10, 10, 200, 400) ;
      r.print(); // r.method() not r->method()
   }
}
```



- Class variable points to the created object !

No-argument Constructor

- Many classes contain a constructor with no arguments that creates an object whose state is set to an appropriate default
  - Numeric values: 0, boolean: false, object variable: null
- If you write a class with no constructors at all, then a no-argument constructor is provided for you.
  - This constructor sets all the instance fields to their default values.

```
public class Employee {
   private int id;
   private String name;
   private double salary;
   public Employee() {
      id = 0;
      name = null;
      salary = 0.0;
   }
}
```

Constructor Chaining

- **Constructor chaining** is the process of calling one constructor from another constructor with respect to current object.
  - **Within same class**: It can be done **using this() keyword** for constructors in same class
  - From base class: by using super() keyword to call constructor from the base class.

```
public class Employee {
   private int id;
   private String name;
   private double salary;
   public Employee() {
      this(0, null, 0.0); //first
   }
   public Employee(int id) {
      this(id, null, 0.0); //first
   }
   public Employee(int id, String name,
double salary) {
      this.id = id;
      this.name = name;
      this.salary = salary;
   }
}
```

https://www.geeksforgeeks.org/constructor-chaining-java-examples/

Class Variable

- Class variable is a **reference** to the created object ! It's not an object.

```
public class Rectangle {
   private Point p ; // Error! It should be Point p = new Point()
   …
   public static void main(String[] args) {
      Rectangle r ; // Error! It should be Rectangle r = new Rectangle() ;
      r.print();
      System.out.println(p) ;
   }
}
```

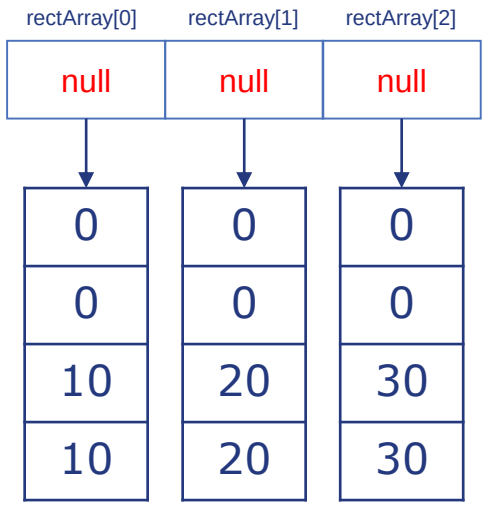
- The program will
  - issue an compile-time error "The local variable r may not have been initialized" or
  - throw an exception "java.lang.NullPointerException"

Array of Objects

- An array of objects stores **a reference variable** of the object in the elements of the array.

```
public class RectangleTest {
    public static void main(String[] args) {
        //Instantiate the array of objects
        Rectangle[ ] rectArray = new Rectangle[3];

        // Initializing Array Of Objects
        rectArray[0] = new Rectangle(0, 0, 10, 10);
        rectArray[1] = new Rectangle(0, 0, 20, 20);
        rectArray[2] = new Rectangle(0, 0, 20, 20);
    }
}
```



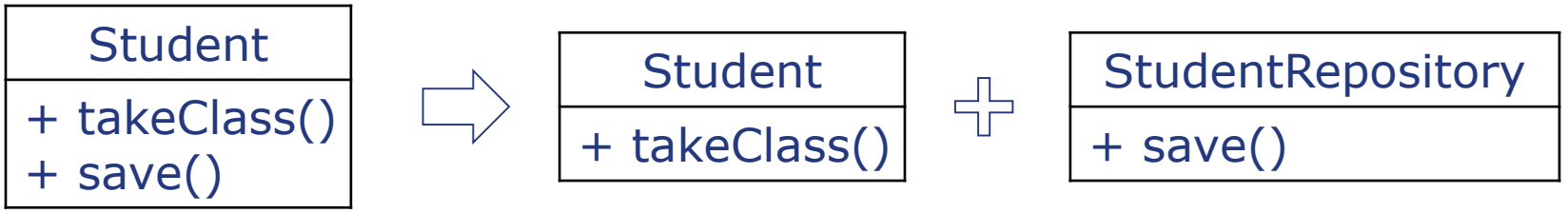
Class: Summary

- Each class can be public or not.
- A class consists of fields(variables) and methods(functions).
- Each field and method can be public, private, or protected.
- All the methods should be implemented within the class.

Good Design:
단일 책임 원칙 (Single Responsibility Principle)

클래스는 단 한 가지의 변경 이유만 가져야 한다!
A class should have only one reason to change.

- 책임 (responsibility)
  - 클래스가 변경되는 이유
  - 클래스를 변경하기 위한 한 가지 이상의 이유가 있다면 그 클래스는 여러가지 책임을 맡고 있다고 판단함
  - 책임을 묶어서 생각하는데 익숙해서 알아 차리기 어려움
  - 변경이 실제로 일어날 때가 진짜 변경이며, 책임을 잘못 분리하면 오히려 경직성(rigidity, 변경이 어려움) 와 취약성(fragility, 맨날 수정함) 가 발생함



- 실제로 객체 지향 디자인 관련 개념은 추상적이라서 많은 훈련이 필요함

Good Design: 책임 (Responsibility)

- 어떤 객체가 어떤 요청에 대해 대답해 줄 수 있거나, 적절한 행동을 할 의무가 있는 경우 해당 객체가 ***책임*** 을 가진다고 함
- 객체의 ***책임*** 은 객체가 알고 있는 것과 할 수 있는 것으로 구성됨
  - 하는 것: 객체를 생성하거나 계산하는 등의 스스로 하는 것, 다른 객체의 행동을 시작시키는 것, 다른 객체의 활동을 제어하고 조절하는 것
  - 아는 것: 자신의 정보, 관련 있는 객체에 관한 정보, 자신이 유도하거나 계산할 수 있는 정보
- 객체의 ***책임*** 을 이야기 할 때는 일반적으로 외부에서 접근 가능한 public 멤버 함수의 관점에서 이야기 함
  - 객체의 외부에 제공해 줄 수 있는 정보 (아는 것) 과 외부에 제공해 줄 수 있는 서비스 (하는 것)의 목록
- 어떤 객체가 수행하는 ***책임*** 의 집합은 객체가 협력 안에서 수행하는 ***역할*** (role) 을 암시함
- ***협력*** (collaboration)에 참여하는 객체들이 너무도 유사하게 협력해서 하나의 협력으로 다루고 싶을 때, ***역할*** (role)을 사용하여 하나의 협력으로 추상화 할 수 있음
  - 동일한 역할을 수행하는 객체들이 동일한 책임의 집합을 수행할 수 있다는 의미

조영호, 객체지향의 사실과 오해, 위키북스, 2015

16

Information Hiding

- Each field and method can be public, private, or protected.
- Only public members can be accessed from outside of the class

```
// Rectangle2.java
class Rectangle2 {
  private int leftTopX, leftTopY ;
  private int rightBottomX, rightBottomY ;
  private void setLeftTop(int x, int y) { leftTopX = x ; leftTopY = y ; }
  private void setRightBottom(int x, int y) { rightBottomX = x ; rightBottomY = y ; }

  public Rectangle2(int x1, int y1, int x2, int y2) {
    setLeftTop(x1, y1) ; setRightBottom(x2, y2) ;
  }
  public int getArea() {
    return (rightBottomX - leftTopX) * (rightBottomY - leftTopX) ;
  }
}
```

Information Hiding

```
// RectangleTest.java
class Rectangle2 { // not public class. Each source file can contain only one public class!
    private int leftTopX, leftTopY ;
    private int rightBottomX, rightBottomY ;
    private void setLeftTop(int x, int y) { leftTopX = x ; leftTopY = y ; }
    private void setRightBottom(int x, int y) { rightBottomX = x ; rightBottomY = y ; }

    public Rectangle2(int x1, int y1, int x2, int y2) { setLeftTop(x1, y1) ; setRightBottom(x2, y2) ; }
    public int getArea() { return (rightBottomX - leftTopX) * (rightBottomY - leftTopX) ; }
}
public class RectangleTest {
    public static void main(String[] args) {
       var r1 = new Rectangle2(0, 0, 50, 50) ;
       var r2 = new Rectangle2(0, 0, 100, 100) ;

       System.out.println(r1.getArea()) ;
       System.out.println(r2.getArea()) ;

       r1.setLeftTop(10, 10) ; // The method setLeftTop(int, int) from the type Rectangle2 is not visible
    }
}
```

Information Hiding – Package-Private

- Package is the default visibility. Package visibility will be discussed later.

```
class Rectangle2 {
    private int leftTopX, leftTopY ;
    private int rightBottomX, rightBottomY ;
    void setLeftTop(int x, int y) { leftTopX = x ; leftTopY = y ; }
    void setRightBottom(int x, int y) { rightBottomX = x ; rightBottomY = y ; }

    public Rectangle2(int x1, int y1, int x2, int y2) {
       setLeftTop(x1, y1) ; setRightBottom(x2, y2) ;
    }
    public int getArea() { return (rightBottomX - leftTopX) * (rightBottomY - leftTopX) ; }
}
public class RectangleTest {
    public static void main(String[] args) {
       var r1 = new Rectangle2(0, 0, 50, 50) ;
       r1.setLeftTop(10, 10) ; // OK
    }
}
```

- Package visibility is very dangerous! **Be sure to specify "private" or "public".** Don't leave it blank.