

Object – Oriented Programming

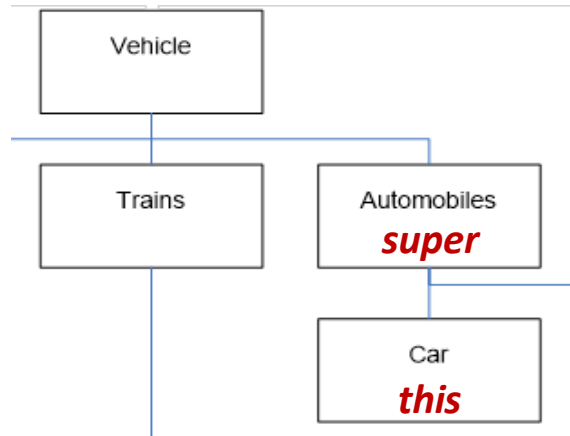
LAB #8. Polymorphism and Abstract Classes

Polymorphism

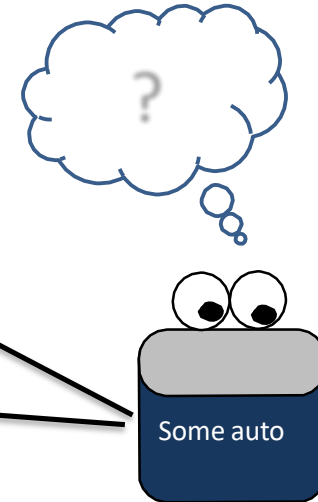
- late binding 메커니즘을 통해 하나의 메소드 이름에 많은 의미를 연결 하는 기능
- late binding 혹은 dynamic binding 이라고 알려진 특별한 메커니즘을 통해 이루어짐

Polymorphism

- Polymorphism은 상속된 클래스의 메소드 정의를 재정의(Override)하며, 이러한 변경 사항을 base class 용으로 작성된 소프트웨어에 적용할 수 있게 함



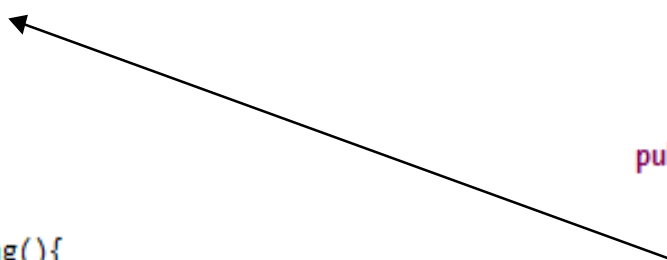
```
public class Automobile{  
    ---  
    public String toString(){  
        ---  
    }  
}  
  
public class Car{  
    ---  
    public String toString(){  
        ---  
    }  
}
```



Binding

- 메소드 호출과 메소드 정의를 연결하는 과정

```
public class Automobile{  
    ---  
    public String toString(){  
        ---  
    }  
}  
  
public class Car{  
    ---  
    public String toString(){  
        ---  
    }  
}  
  
public static void main (String[] args){  
    Automobile auto = new Automobile();  
    System.out.println(auto.toString());  
}
```

A black arrow originates from the `auto.toString()` call within the `main` method of the `Car` class and points to the `toString()` method definition inside the `Automobile` class, illustrating the binding process.

Binding

- Early binding (static binding)
 - 코드가 컴파일 될 때, 메소드 정의가 메소드 호출과 연결됨
 - Late binding (dynamic binding)
 - 메소드가 런타임에 호출될 때, 메소드 정의가 메소드 호출과 연결됨
 - Java는 모든 메소드에 대하여 late binding을 사용한다. (final과 static은 예외)
-
- Compile time : 코드 작성시
 - Run time : 실행시
 - new 연산자를 통해 객체 생성 후 객체를 통해 호출되는 메소드들은 런타임에서 메소드 정의와 바인딩 되지만 static과 final의 경우 객체 생성없이 컴파일 타임에 바인딩 되므로 late binding이 아님

No Late Binding for Static Methods

- 컴파일 타임에서 사용할 메소드의 정의를 결정할 때, 이를 static binding 혹은 early binding 이라 한다.
 - 객체를 이름 지은 변수의 타입에 기반하여 결정됨
- Java는 private, final, static method에 late binding이 아닌 static binding을 사용한다.
 - Private, final 메소드의 경우, late binding은 어떠한 목적도 제공하지 않음
 - 객체를 이용하여 static method를 호출하는 경우 binding의 종류에 따라 결과가 달라짐

The *final* Modifier

- `final`로 선언된 메소드는 상속된 클래스에서 새로운 정의로 오버라이딩 될 수 없다.
 - > `final`의 경우, 컴파일러는 해당 메소드에 early binding을 함
- `final`로 선언된 클래스는 다른 클래스를 상속하는 base class로 사용될 수 없음

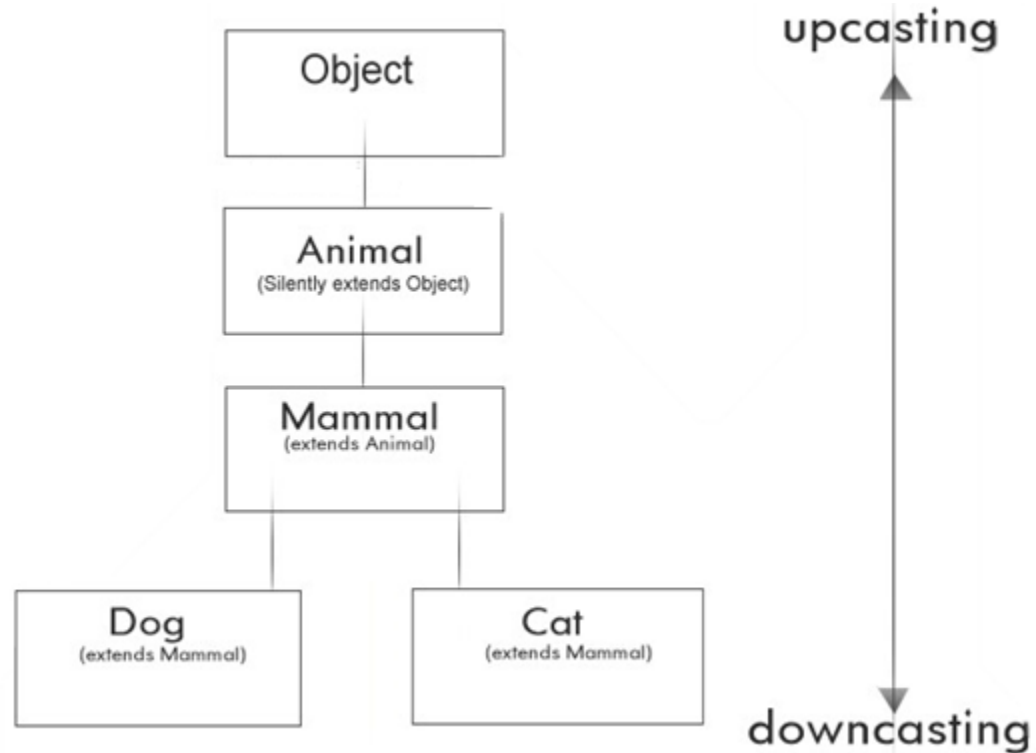
Upcasting and Downcasting

- Upcasting
 - 상속된 클래스의 객체가 base class 혹은 ancestor class의 변수에 할당될 때
- Downcasting
 - base class가 상속된 클래스로 타입 캐스팅이 수행될 때
 - ancestor class가 그 클래스의 하위 클래스로 타입 캐스팅이 수행될 때

Upcasting and Downcasting

```
Cat c = new Cat();  
Mammal m = c; // upcasting
```

```
Cat c1 = new Cat();  
Animal a = c1; // automatic upcasting to Animal  
Cat c2 = (Cat) a; // manual downcasting back to a Cat
```



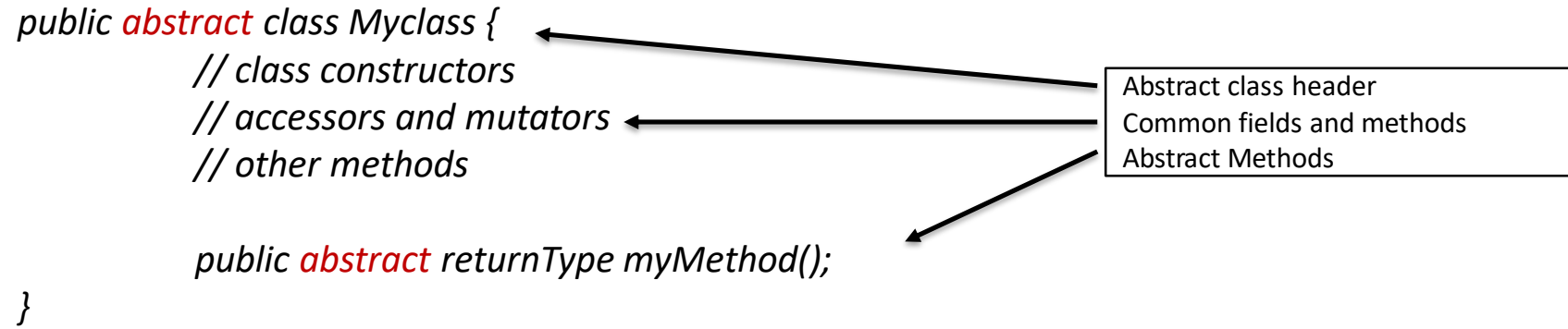
instanceof

- 특정 객체가 무슨 타입인지 알려주는 연산자
 - [객체] instanceof [클래스]
 - [sub class의 객체] instanceof [super class] == true
 - [super class의 객체] instanceof [sub class] == false

Abstract Classes

- 하나 이상의 Abstract Method(추상 메소드)를 포함한 클래스
- abstract method : 완전한 정의가 없는 메소드 (단순한 placeholder)
 - > 선언만 되어있고 구현부가 비어있음
- concrete class : 어떠한 abstract method도 포함하지 않는 클래스
 - > 인스턴스화 가능 (객체를 만들 수 있음)

Defining Abstract Class



When to use

- 밀접하게 관련된 클래스들이 코드를 공유하게 만들고 싶은 경우
- Abstract class를 확장하는 클래스가 여러 개의 공통 메소드 혹은 필드를 가지고 있거나 public이 아닌 다른 Access modifier를 요구할 수 있을 경우(protected)
- non-static 혹은 non-final 필드를 원할 경우, 이를 통해 non-static 혹은 non-final 필드가 속한 객체에 접근하고 수정할 수 있는 메소드를 정의할 수 있음

실습

2차 주

- Employee, Engineer, Manager Class를 생성한다.

- Employee class는 추상클래스로 선언한다.
- Employee class에는 5개의 instance가 있다
 - ◎ String name
 - ◎ int employeeNum
 - ◎ String department
 - ◎ int workHrs;
 - ◎ double salary;
- name, employeeNum를 받는 생성자를 작성한다. -> workHrs, salary는 0으로 초기화
- department에 대한 getter와 setter를 작성하고, workHrs에 대한 getter를 작성한다.
- Employee의 name과 employeeNum이 같으면 true를 반환하는 equals(Object obj) method를 작성한다. *override*
- Employee의 name, employeeNum를 String으로 반환하는 toString()을 작성한다.
- 일한 시간(workHrs)을 hrs 만큼 증가시키고 getPaid()를 호출하여 salary를 갱신하는 doWork(int hrs)를 작성한다.
- Abstract method getPaid()을 작성한다. (return : double형)
- Employee의 salary가 같으면 true를 반환하는 equalPay(Employee emp)를 작성한다.

실습


- Engineer class 는 Employee class를 extends 한다.
 - Engineer class에는 1개의 instance가 있다.
 - ◎ int rate
 - name, employeeNum을 인자로 받는 생성자를 작성한다.
 - > rate 는 4.0 으로 초기화, department : Engineering 설정
 - Engineer의 name과 employeeNum이 같으면 true를 반환하는 equals(Object obj)를 작성한다.
 - Engineer의 name, employeeNum, department를 String으로 반환하는 toString()을 작성한다.
 - pay를 반환하는 getPaid()를 구현한다. ($pay = workhrs * rate$)

실습

- Manager class는 Employee class를 extend한다
 - Manager class에는 3개의 instance가 있다.
 - ◎ double overtimeRate
 - ◎ double rate
 - ◎ int regularHrs
 - name, employeeNum을 인자로 받는 생성자를 작성한다.
 - > rate = 5.0 , overtimeRate = 8.0, regularHrs = 40 으로 초기화, department : Management 설정
 - Manager의 name과 employeeNum이 같으면 true를 반환하는 equals(Object obj)를 작성한다.
 - Manager의 name, employeeNum, department를 String으로 반환하는 toString()을 작성한다.
 - pay를 반환하는 getPaid()를 구현한다.
 - $\text{int overtimeHrs} = \text{workHrs} - \text{regularHrs}$
 - WorkHrs이 40시간 미만일 경우, $\text{Pay} = \text{workHrs} * \text{rate}$
 - WorkHrs이 40시간 이상일 경우, $\text{Pay} = (\text{regularHrs} * \text{rate}) + (\text{overtimeHrs} * \text{overtimeRate})$

실습

- Class Company를 생성하고 다음과 같이 main method를 작성 후 출력 결과를 확인해본다.



```
public class Company {
    public static void main(String[] args) {
        Employee emp1 = new Manager("John Smith", 1234);
        Employee emp2 = new Engineer("Peter Anderson", 1432);
        Manager emp5 = new Manager("Jenny Allen", 1734);

        System.out.println(emp1.toString());
        System.out.println(emp2.toString());
        System.out.println(emp5.toString());

        emp1.doWork(20);
        emp2.doWork(90);

        System.out.println("emp1 Salary: " + emp1.getPaid());
        System.out.println("emp2 Salary: " + emp2.getPaid());
        System.out.println("emp5 Salary: " + emp5.getPaid());

        System.out.println("emp1 and emp2 has the same salary? " + emp1.equalPay(emp2));
        System.out.println("emp1 and emp5 has the same salary? " + emp1.equalPay(emp5));
        emp5.doWork(20);
        System.out.println("emp1 and emp5 has the same salary? " + emp1.equalPay(emp5));
        emp1.doWork(40);
        System.out.println("emp1 and emp2 has the same salary? " + emp1.equalPay(emp2));
    }
}
```

```
Name : John Smith
Emp# : 1234
Dept : Management
```

```
Name : Peter Anderson
Emp# : 1432
Dept : Engineering
```

```
Name : Jenny Allen
Emp# : 1734
Dept : Management
```

```
emp1 Salary: 100.0
emp2 Salary: 360.0
emp5 Salary: 0.0
emp1 and emp2 has the same salary? false
emp1 and emp5 has the same salary? false
emp1 and emp5 has the same salary? true
emp1 and emp2 has the same salary? true
```

6주차 실습 과제 관련 질문: minah741@naver.com