Lab 4

ADT Design

- 클래스로 정의됨.
- 모든 객체들은 힙 영역에 할당됨.
- 캡슐화(Encapsulation) : Data representation + Operation
- 정보은닉(Information Hiding) : Opertion부분은 가려져있고, 사용자가 operation으로만 사용 가능해야 함.

클래스 정의의 형태

```
public class Person {

    private String name;
    public int age;
    public Person() {
    }
    public Person(String s) {
        name = s;
    }
    public String getName() {
        return name;
    }
}
```

생성자 :

클래스의 이름과 동일한 메소드, 클래스의 객체가 생성될 때 호출되는 메소드

메소드 :

실행 가능한 함수, 객체의 행위를 구현

메소드 오버로딩 :

① 메소드 이름 동일

② 메소드 인자의 개수가 서로 다르거나, 메소드 인자의 타입이 서로 달라야 함.

③ 리턴 타입만 다른 경우에는 에러

객체의 생성

```
public static void main (String args[]) {

        Person aPerson;
        aPerson = new Person("홍길동");
        aPerson.age = 30;
        String s = aPerson.getName();

}
```

객체에 대한 레퍼런스 변수

aPerson을 선언한 뒤, new를 통해서

Person 객체를 생성하고 있다.

멤버 접근 지정자

멤버에 접근하는 클래스	멤버의 접근 지정자			
	default	private	protected	public
같은 패키지의 클래스	O	X	O	O
다른 패키지의 클래스	X	X	X	O

멤버 접근 지정자

- public인 name에 대해서는 바로 값 지정이 가능하지만, private으로 선언 된 age의 경우 바로 선언하려고 하면 에러가 발생한다.
- 이는 private의 경우, 서로 다른 클래스에 선언되어 있기 때문이다. 때문에 클래스 내부에서 get/set 메소드를 만들어서 접근해야 한다.

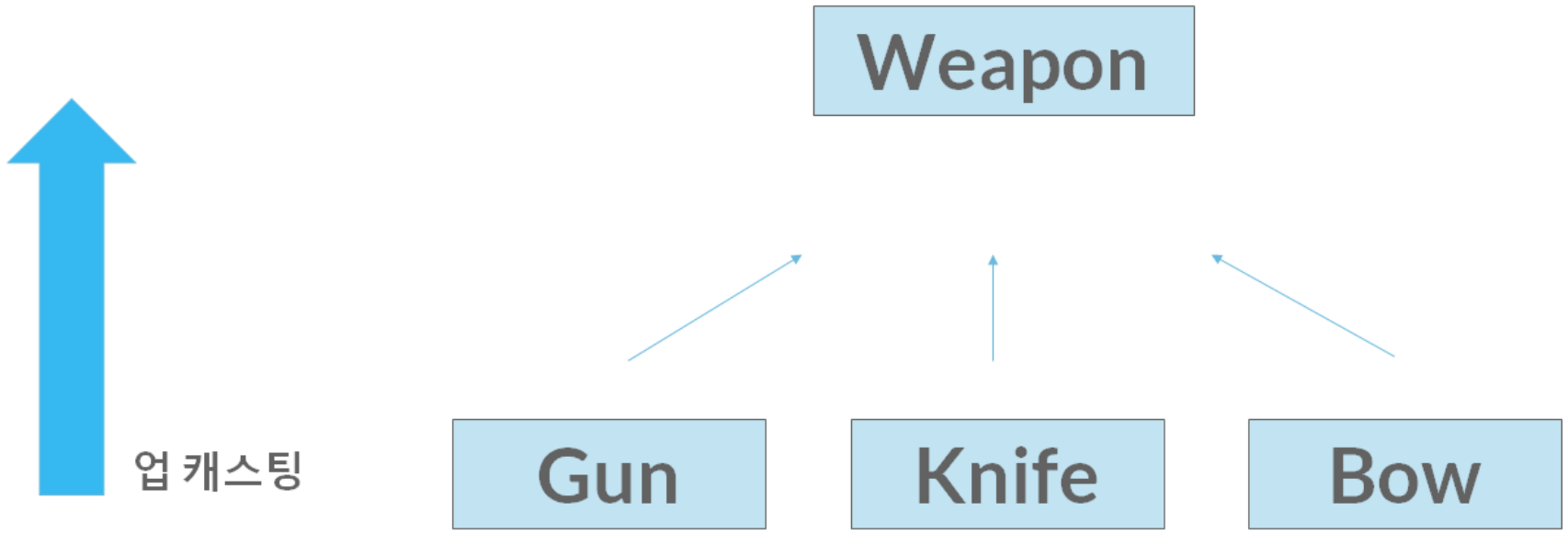
```
class Person {
	public String name;
	private int age;
	int num;

	public int getAge() {
		return age;
	}
	public void setAge(int value) {
		age = value;
	}
}
public class Access {
	public static void main(String[] args) {
		Person aPerson = new Person();
		aPerson.name = "홍길동";
		aPerson.setAge(20);
		aPerson.num = 10;
	}
}
```

Class Hierarchy - Casting

- 객체의 타입 변환
- 업캐스팅 : 서브클래스 객체가 슈퍼 클래스 타입으로 변환되는 것
- 다운캐스팅 : 업캐스팅 된 것을 다시 원래대로 되돌리는 것, 명시적으로 타입 지정이 필요.

Class Hierarchy - Casting



Class Hierarchy - Casting

- C++ 의 경우
- GUN을 weapon으로 바꾸었다기 보다는 메모리에는 GUN 고대로 존재하되 (동적할당)
- 그 시작점을 가리키는 포인터를 부모로 가리키겠다는 의미이다.

```
Weapon* w = new Gun();
또는
Weapon* w;
Gun* g = new Gun();
w = (Weapon*)g;
```

```
class Person {
	String name;
	String id;
	public Person(String name)
	{ this.name = name;
}
class Student extends Person {
	String grade;
	String department;
	public Student(String name) {
		super(name);
	}
}
public class Casting {
	public static void main(String[] args) {
		Person p = new Student("홍길동"); // ① 업캐스팅
		System.out.println(p.name);
		p.grade = "A";
		p.department = "컴퓨터";
		Student s = (Student)p; // ② 다운캐스팅
		System.out.println(s.name);
		s.grade = "A";
		}
}
```

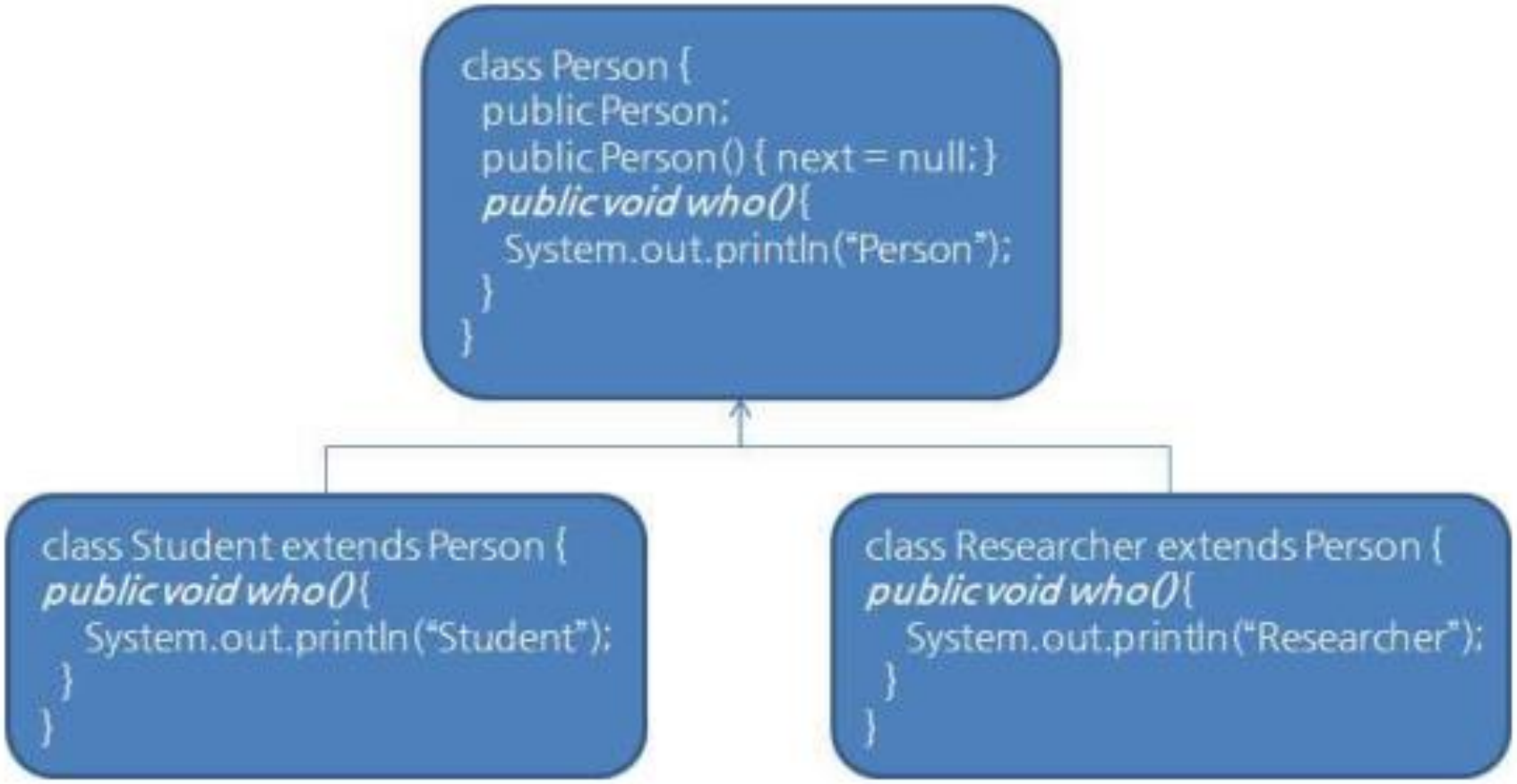
Class Hierarchy - Overriding

- 슈퍼클래스와 서브 클래스의 메소드 사이에 발생하는 관계.
- 슈퍼클래스의 메소드를 동일한 이름으로 서브 클래스에서 재 작성하는 것.
- super 키워드를 통해서 슈퍼 클래스에 대한 멤버와 메소드에 접근 가능.

Class Hierarchy - Overriding

- 오버라이딩의 조건
- ① 슈퍼 클래스의 메소드와 완전히 동일한 메소드를 재정의.
- ② 슈퍼 클래스 메소드의 접근 지정자보다 접근의 범위가 좁아질 수 없음.
- ③ 리턴 타입만 다를 수 없음.

Class Hierarchy - Overriding



Class Hierarchy - Overriding

```
public class Overriding {

	public static void main(String[] args) {
		Person p = new Person();
		Student st = new Student();
		Person p1 = new Researcher();
		Person p2 = st;
		p.who();
		st.who();
		p1.who();
		p2.who();
	}
}
```

출력결과 : Person / Student / Researcher / Student

Student와 Researcher의 who() 메소드는 오버라이딩 된 형태이다.

그렇기 때문에 자식 클래스에 있는 함수가 실행된다.

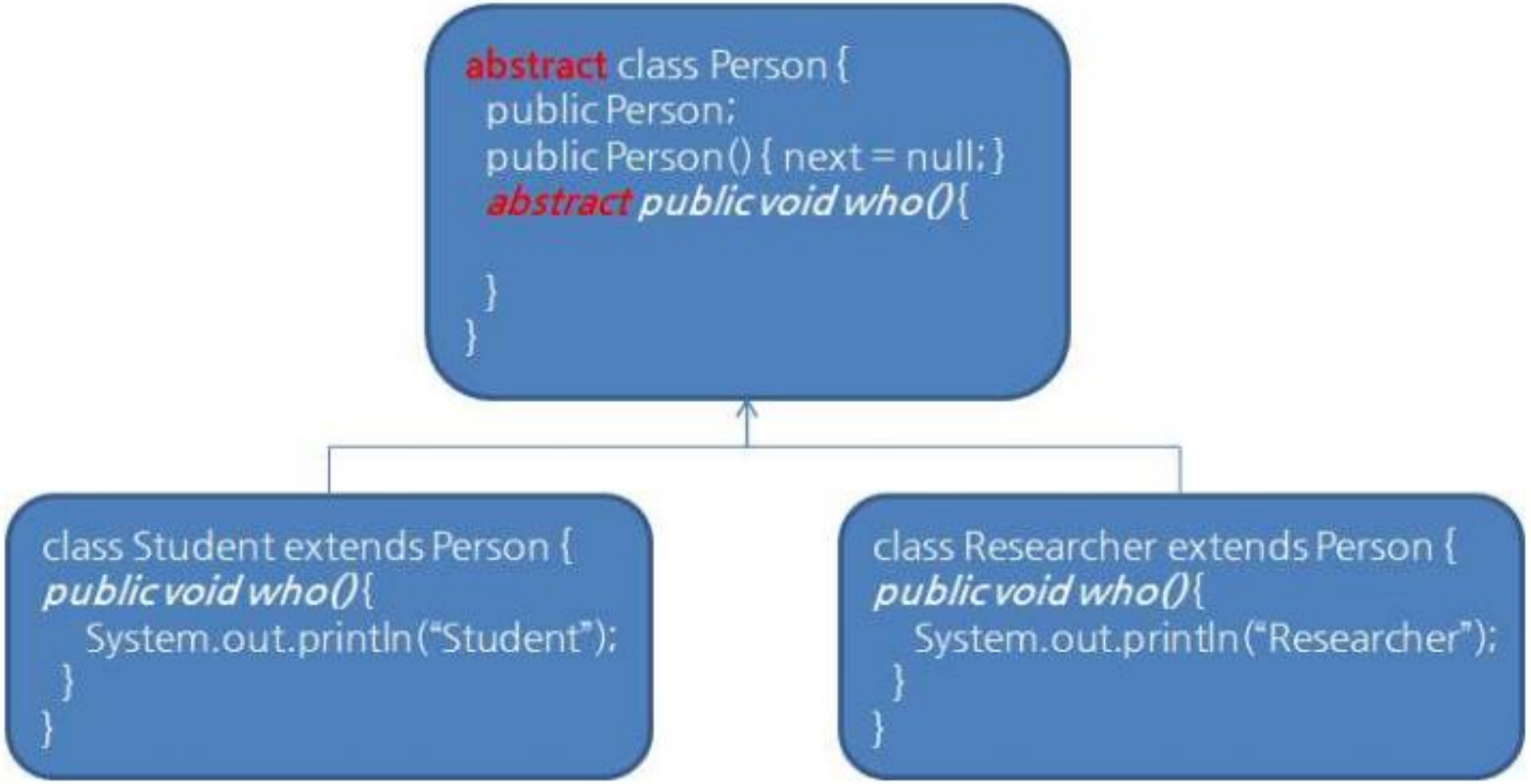
Method Overloading vs Overriding

	Overloading	**Overriding**
정의	같은 클래스나 상속 관계에서 동일한 이름의 메소드 중복 작성	서브 클래스에서 슈퍼 클래스에 있는 메소드와 동일한 이름의 메소드 작성
관계	같은 클래스나 상속 관계	상속 관계
목적	이름이 같은 여러 개의 메소드를 중복 정의하여 사용의 편리성 향상	서브 클래스에서 새로운 기능의 메소드를 재정의
조건	메소드 이름 동일, 메소드의 인자의 개수나 인자의 타입이 달라야 함.	메소드의 이름, 매개변수의 형태 등이 모두 동일
바인딩	정적 바인딩	동적 바인딩

Abstract Class

- 추상클래스의 상속
  - 추상 클래스를 상속받으면 상속받은 서브 클래스는 추상 클래스가 됨.
  - 서브 클래스가 추상 클래스가 되지 않기 위해서는 추상 메소드를 모두 오버라이딩해야 함.

Interfaces – Abstract Class



Interfaces

- 추상클래스의 한 종류로, 다중 상속이 가능하게 함.
- implements 키워드는 인터페이스의 추상 메소드를 클래스에서 구현하는것을 말함.

Interfaces

- 인터페이스의 특징
  - ① 추상클래스와 다르게 반드시 추상 메소드와 상수만으로 구성
  - ② 모든 메소드는 abstract public이며 생략 가능함.
  - ③ 상수도 public static final을 생략하여 선언 가능함.
  - ④ 인터페이스의 객체를 생성할 수 없음.
  - ⑤ 다른 인터페이스에 상속될 수 있음.
  - ⑥ 인터페이스도 레퍼런스 변수의 타입으로 사용 가능.

```
interface CanFight {
void fight();
}
interface CanSwim {
void swim();
}
interface CanFly {
void fly();
}
class ActionCharacter {
public void fight() {
System.out.println("fight");
}
}
class Hero extends ActionCharacter implements
CanFight, CanSwim, CanFly {
public void swim() {
System.out.println("swim");
}
public void fly() {
System.out.println("fly");
}
}
```

```
public class Adventure {
public static void t(CanFight x) {
x.fight();
}
public static void u(CanSwim x) {
x.swim();
}
public static void v(CanFly x) {
x.fly();
}
public static void w(ActionCharacter x) {
x.fight();
}
public static void main(String[] args) {
Hero h = new Hero();
t(h); // Treat it as a CanFight
u(h); // Treat it as a CanSwim
v(h); // Treat it as a CanFly
w(h); // Treat it as an ActionCharacter
}
}
```

```
interface Monster {
void menace();
}
interface DangerousMonster extends Monster {
void destroy();
}
interface Lethal {
void kill();
}
class DragonZilla implements DangerousMonster {
public void menace() {
System.out.println("menace1");
}
public void destroy() {
System.out.println("destory1");
}
}
```

```
interface Vampire extends DangerousMonster, Lethal {
void drinkBlood();
}
class VeryBadVampire implements Vampire {
public void menace() {
System.out.println("menace2");
}
public void destroy() {
System.out.println("destroy2");
}
public void kill() {
System.out.println("kill2");
}
public void drinkBlood() {
System.out.println("drinkBlood2");
}
}
```

```
public class HorrorShow {
    static void u(Monster b) {
    b.menace();
    }
    static void v(DangerousMonster d) {
    d.menace();
    d.destroy();
    }
    static void w(Lethal l) {
    l.kill();
    }
    public static void main(String[] args) {
        DangerousMonster barney = new DragonZilla();
        System.out.println("1");
        u(barney);
        System.out.println("2");
        v(barney);
        Vampire vlad = new VeryBadVampire();
        System.out.println("3");
        u(vlad);
        System.out.println("4");
        v(vlad);
        System.out.println("5");
        w(vlad);
    }
}
```

인터페이스를 만들 때, 다중 인터페이스 상속이 가능함.
인터페이스를 통한 다중 상속이 구현되었음을 확인 할 수 있음.

인터페이스 vs 추상클래스

추상클래스	인터페이스
- 일반 메소드 포함 가능 - 상수, 변수 필드 포함 가능 - 모든 서브 클래스에 공통된 메소드가 있는 경우에는 추상 클래스가 적합	- 모든 메소드가 추상 메소드 - 상수 필드만 포함 가능 - 다중 상속 지원