[과제] 상속과 인터페이스 (수강생용)

과제 개요

1. **버거왕**의 버거 메뉴를 구현해 보자
2. 햄버거 세트 메뉴 구현해보자
3. 버거를 만드는 아르바이트 생을 만들자

과제 목표

1. 상속을 구현해 본다.
2. 상속과 포함 이해

(가지고 있다/ 이다 / is-a / has-a / 고래 / 포유류...@#$)

3. super와 super() 이해하기
4. 인터페이스 이해

요구사항

1. 버거왕을 만들었다. 앞으로 여러가지 버거를 만들기 위해 버거 클래스를 만들자.

```
버거 클래스 {
    * 패티(patty): 정수형
    * 야채(vegetables): 문자열 리스트
    * 추가 재료(extras): 문자열 리스트
}
```

2. 먹을 수 있는 실제 버거, 와퍼를 만들어 보자
   와퍼는 특별한 글레이즈드 번을 쓴다고 한다.

▼ 와퍼

```
public class 와퍼 <_______> 버거 {
    String bunOption;

    public 와퍼 생성자() {
        <_______>
        this.bunOption = "GLAZED";
    }

}
```

▼ 퀴즈

💡 <Quiz> 여기서 버거와 와퍼의 관계는?
1. 버거 is a 와퍼
2. 와퍼 is a 버거

3. 버거를 만들려면 버거를 만드는 레시피가 있어야 할 것 같다.

▼ 그런데 레시피는 버거마다 다를텐데..?

```
public class Burger {
    int patty;
    List<String> vegetables;
    List<String> extras;
```

```
    public <________> Burger recipe()<__>
}
```

▼ 그럼 와퍼 레시피를 만들어야겠다.
와퍼는 110g의 패티를 특별하게 구워서 만든다고 한다.
그리고 야채에는 토마토(Tomato)/양상추(Lettuce)/양파(Onion)/피클(Pickle)이 들어간다

```
public class 와퍼 /*... 버거 */ {
    String bunOption;

    public 와퍼 생성자() {
        // ..
    }

    // 와퍼 레시피 구현하기
    <________>
    public <_______> recipe() {
        // 패티를 특별하게 굽기
        // 야채를 추가한다. 야채는 어딨지?
        // 레시피를 만들고 나면 나오는 것은 와퍼일까 버거일까?
    }

    // 패티 110g을 특별하게 굽는 메소드 구현하기
    public void grillPatty() {
        패티 = 110;
    }
}
```

4. 버거만 먹으면 목이 메이니 콜라를 만들자! 그런데 콜라 말고 다른 **음료**도 곧 추가될 것 같은데?

▼ 콜라? 음료?

```
public class 음료 {
    String name;
    String size;
}

public class 콜라 <_______> 음료 {


}
```

▼ 퀴즈

💡 <Quiz> 여기서 음료와 콜라의 관계는?
1. 음료 is a 콜라
2. 콜라 is a 음료
3. 콜라 has a 음료
4. 음료 has a 콜라

5. 매출을 높이려면 버거와 음료를 묶어서 세트로 만들어야겠다. 세트 메뉴를 만들자!

```
public class SetMenu {
    버거;
    음료;

    // 버거와 음료를 하나씩 갖는 생성자 구현
}
```

6. 이제 직원을 고용해야할 것 같다.

▼ 직원에게 무슨 **역할**을 맡길까?

```
public <_________> StaffRole {
    // 버거를 만든다
    // 음료를 뽑는다
    // 세트를 만든다
    // 주문을 받는다
}
```

▼ 주문량이 늘어서 캐셔와 주방 직원의 역할을 분리하자

```
public interface StaffRole {
    default void recordAttendance() {
        System.out.println("출근");
    }

    default void recordLeaving() {
        System.out.println("퇴근");
    }
}



public <_________> CashierRole <_________> StaffRole {
    void serveFood(String servingType);

    // 주문을 받는다
    // 계산한다

}

public <_________> CookerRole <_________> StaffRole {
    // 버거를 만든다
    // 음료를 뽑는다
}
```

▼ 운영시간이 늘어나며 직원들을 파트타임으로 고용했다.
모든 직원들은 출근시간과 퇴근시간을 기록해야 한다.

```
public <_________> CashierRole {
    // 주문을 받는다
    // 계산한다

    // 출근시간을 기록한다
    // 퇴근시간을 기록한다
}

public <_________> CookerRole {
    // 버거를 만든다
    // 음료를 뽑는다

    // 출근시간을 기록한다
    // 퇴근시간을 기록한다
}
```

공통된 행동을 하나로 묶을 수 없을까?

```
public <_________> StaffRole {
    // 출근시간을 기록한다
    // 퇴근시간을 기록한다
}


public <_________> CashierRole <_________> StaffRole{
    // 주문을 받는다
    // 계산한다
}

public <_________> CookerRole <_________> StaffRole {
    // 버거를 만든다
    // 음료를 뽑는다
}
```

만약 둘 다 책임지는 매니저를 고용한다면?

```
public <_________> ManagerRole <_________> CookerRole, Cash

}
```

▼ 주방 파트타임 스탭을 고용하자

```
public class KitchenPartTimer <_________> CookerRole{
// 출근하면 출근 카드를 기록하는 startWork()를 구현
// 퇴근할 때 퇴근을 기록하는 endWork()를 구현

// ...
}
```

▼

```
public class KitchenPartTimer implements CookerRole{
        * 이름 - 문자열
        * 급여 - 정수형

    public void startWork() {
        System.out.println("주방 파트타이머 출근");
                // 출근을 기록하는 부모 클래스의 메소드 수행
    }

    public void endWork() {
        System.out.println("주방 파트타이머 퇴근");
        // 퇴근을 기록하는 부모 클래스의 메소드 수행
    }

        <_________>
    public Burger makeBurger(Burger burger) {
        System.out.println("파트타이머가 버거를 만듭니다");
```

```
        return burger;
    }

        <_________>
    public void completeOrder(Burger completeBurger, Dri
        System.out.println("만든 음식을 서버에게 전달합니다");
    }
}
```