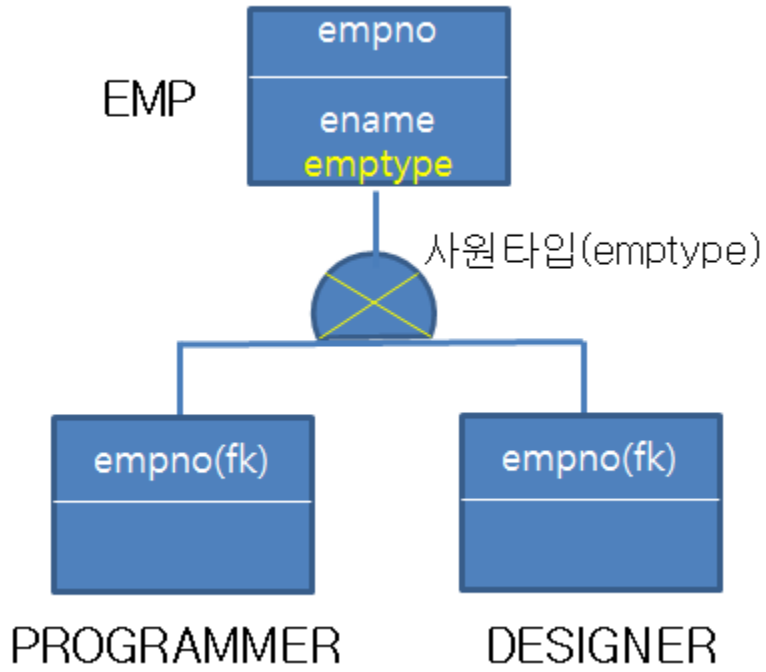
## 상속 관계 매핑

- RDB는 객체지향처럼 상속이 존재하지 않으며 대신 Super Type, Sub Type 이라는 개념이 존재한다.
- 슈퍼/서브 타입 논리모델을 물리적인 테이블로 변환할 때는 하나의 통합된 테이블로 표시(단일 테이블 전략)하거나, 각각 테이블로 별도로 두어 조회할 때 조인을 이용하거나 서브타입만 테이블로 변환을 하는 방법이 있다.




**1. 상속관계 매핑(별도 테이블 조인 전략)**

엔티티를 각각 테이블로 만들고 부모/자식 관계를 이용하여 표현하며 부모 쪽에 자식들의 타입을 구분하는 칼럼(emptype)이 추가 되어야 한다.

```
@Entity
@Table(name="EMP")
@Inheritance(strategy=InheritanceType.JOINED)
@DiscriminatorColumn(name="EMPTYPE", discriminatorType=STRING, length=1)
public abstract class Emp {
}

@Entity
```

```
@Table(name="PROGRAMMER")
@DiscriminatorValue(value="P")
@PrimaryKeyJoinColumn(name="EMPNO")
public class Programmer extends Emp { ...    }

@Entity
@Table(name="DESIGNER")
@DiscriminatorValue(value="D")
@PrimaryKeyJoinColumn(name="EMPNO")
public class Designer extends Emp {    }
```

**2. 상속관계 매핑(단일 테이블 전략)**

- 상속 계층의 모든 테이블을 한 테이블로 통합하는 것을 말하며 구분칼럼(Discriminator Column)으로 객체를 구별하므로 반드시 필요하다.
- 조인이 필요 없으므로 일반적으로 빠르지만 칼럼들이 널 값을 많이 가질 수 있다.
- @DiscriminatorColumn를 반드시 설정해야 하고 @DiscriminatorValue를 지정하지 않으면 엔티티 이름을 기본으로 사용한다.

```
@Entity
@Table(name="EMPS")
@Inheritance(strategy=InheritanceType.SINGLE_TABLE)
@DiscriminatorColumn(name="EMPTYPE",      discriminatorType=DiscriminatorType.STRING,
length=1)
public abstract class Emp {   }

@Entity
@DiscriminatorValue(value="P")
public class Programmer extends Emp { }
@Entity
@DiscriminatorValue(value="D")
public class Designer extends Emp {   }
```

**3. 상속관계 매핑(구현 클래스마다 테이블 전략)**

- 자식 엔티티(PROGRAMMER, DESIGNER)들을 테이블로 만드는 방법으로 바람직하지는 않다.

```
@Entity
```

```
//@MappedSuperclass 어노테이션도 동일한 역할을 한다. 이 경우 Emp 엔티티는  DB에 생성 안됨
@Inheritance(strategy=InheritanceType.TABLE_PER_CLASS)
public abstract class Emp {  }

@Entity
@Table(name="PROGRAMMER")
public class Programmer extends Emp {  }

@Entity
@Table(name="DESIGNER")
public class Designer extends Emp {  }
```