추상 클래스와 인터페이스

- 추상 클래스
- 인터페이스

9-1

요구사항

- 직원관리 프로젝트를 진행하려고 한다.
- 직원은 아르바이트, 정규직으로 구분합니다.
- 정규직은 일반 사원과 팀장으로 구분합니다.
- 팀장은 전용 방과 전화번호가 있다.
- 정규직은 소속 부서와 입사 년도와 연봉이 있다.
- 아르바이트는 계약 기간과, 계약시 시급 정보를 관리합니다.

- 위 조건을 만족하는 클래스 다이어그램과 멤버를 선언하세요.

9-2

추상 클래스와 인터페이스

- 추상 클래스
- 인터페이스
- class, abstract class, interface

9-3

추상 클래스

- 추상 메서드는 구현부가 없고 이름만 있는 메서드
- 추상 메서드를 1개라도 가진 클래스는 무조건 추상 클래스

```
abstract class AbstractClass{

   abstract public void print();
   public void methodA(){
       System.out.println("SuperClassA's methodA()");
  }
}
```

9-4

추상 클래스

- 추상 클래스를 상속받은 클래스는 메서드를 재정의 해야한다.

```
class SubClassA extends AbstractClass{
  @Override
  public void print() {
       System.out.println("SubClassA's  print()");
  }
}
```

- 추상 클래스를 상속받은 클래스도 추상 클래스로 정의한다.

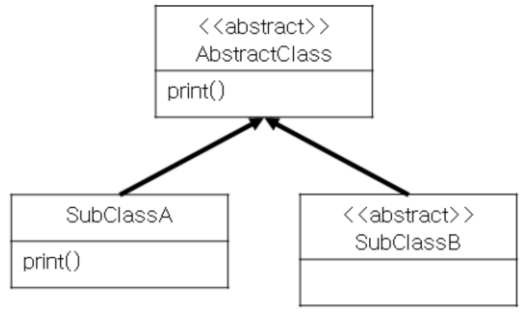
```
abstract class SubClassB extends AbstractClass{
}
```

9-5

추상 클래스

- 추상 클래스는 생성자는 있지만 new 뒤에서 사용 불가
- 자식 클래스의 생성자를 통해서만 생성이 가능

```
AbstractClass s1=new AbstractClass();  //컴파일 오류
AbstractClass s2=new SubClassA();
AbstractClass s3=new SubClassB();      //컴파일 오류
```



9-6

추상 클래스

- 추상 메서드는 없지만 추상 클래스로 선언 가능
- 상속을 유도

```
abstract class AbstractClassC{
  public void methodC(){
      System.out.println("SuperClassC's  methodC()");
  }
}
class SubClassC extends AbstractClassC{}
```

- 추상 메서드가 없어도 추상 클래스면 직접 생성 불가능

```
AbstractClassC  s=new AbstractClassC();  //컴파일 오류
AbstractClassC  s=new SubClassC();
s3.methodC();
```

9-7

• 추상 클래스

```
abstract class AbstractClass{
    abstract public void print();//
    public void methodA(){
        System.out.println("AbstractClass's methodA()");
    }
}

class SubClassA extends AbstractClass{
    @Override
    public void print() {
        System.out.println("SubClassA's print()");
    }
}

abstract class SubClassB extends AbstractClass{         }

abstract class AbstractClassC{
    public void methodC(){
    System.out.println("SuperClassC's methodC()");
    }
}
class SubClassC extends AbstractClassC{}

public class AbstractTest {
    public static void main(String[] args) {
        //AbstractClass s1=new AbstractClass();  //
        AbstractClass s2=new SubClassA();
        //AbstractClass s3=new SubClassB();

        //AbstractClassC s3=new AbstractClassC();
        AbstractClassC s3=new SubClassC();

        s2.print();
        s3.methodC();
    }
}
```

추상 클래스

- 추상 클래스의 멤버가 모두 추상 메서드면 모두 재정의

```
abstract class AbstractClassD{
  abstract void methodA();
  abstract void methodB();
  abstract void methodC();
}
class SubClassD extends AbstractClassD{
  public void methodA(){}
  public void methodB(){}
  public void methodC(){}
}
```

- 모든 멤버가 추상 메서드일 경우 다중 상속이 가능한 인터페이스로 정의

9-9

인터페이스

- 모든 멤버가 추상 메서드일 경우 다중 상속이 가능한 인터페이스로 정의
  - abstract class → interface
  - 메서드에 abstract 생략

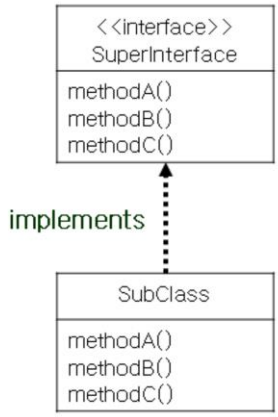
```
abstract class AbstractClass{
    abstract void methodA();
    abstract void methodB();
    abstract void methodC();
}
```

```
interface SuperInterface{
    void methodA();
    void methodB();
    void methodC();
}
```

9-10

인터페이스

- 클래스가 인터페이스를 속 받을 때는 implements 사용

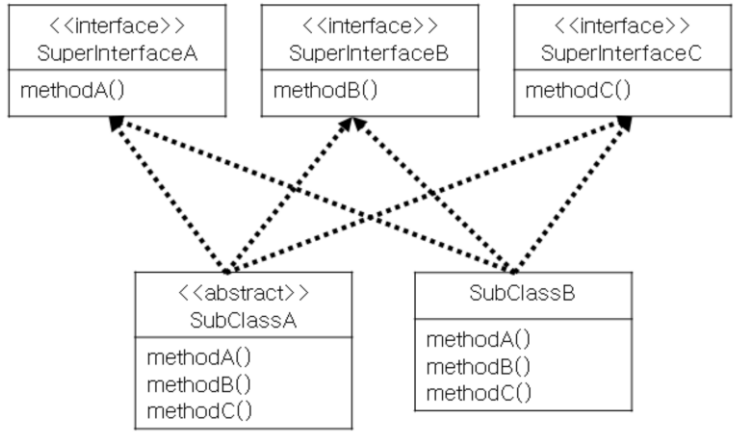



```
interface SuperInterface{
    void methodA();
    void methodB();
    void methodC();
}

class SubClass implements SuperInterface{
    public void methodA(){}
    public void methodB(){}
    public void methodC(){}
}
```

9-11

인터페이스

- 인터페이스는 다중 상속이 가능하다.
- 자식 클래스는 모든 메서드를 재정의 또는 추상 클래스로 정의



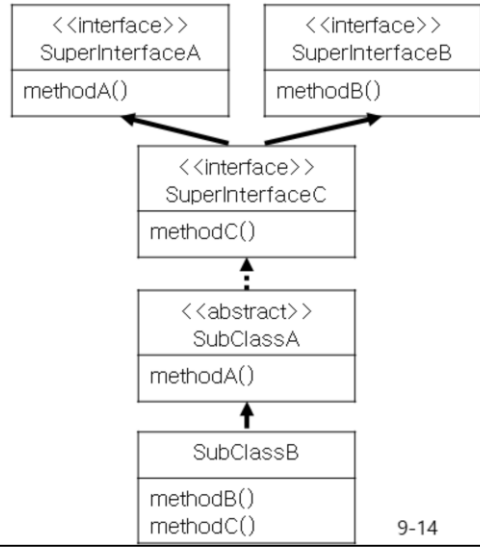
9-12

• 인터페이스

```
interface SuperInterfaceA{
    void methodA();
}
interface SuperInterfaceB{
    void methodB();
}
interface SuperInterfaceC{
    void methodC();
}

abstract class SubClassA implements SuperInterfaceA,
SuperInterfaceB, SuperInterfaceC{

}

class SubClassB implements SuperInterfaceA, SuperInterfaceB,
SuperInterfaceC{

    @Override
    public void methodC() {
        //
    }
    @Override
    public void methodB() {
        //
    }
    @Override
    public void methodA() {
        //
    }
}
```

인터페이스

- 인터페이스끼리 상속 가능
- 클래스 중 일부 구현부가 같은 클래스는 추상 클래스로 정의



9-14

• 인터페이스

```
interface SuperInterfaceA{
    void methodA();
}
interface SuperInterfaceB{
    void methodB();
}
interface SuperInterfaceC extends SuperInterfaceA, SuperInterfaceB{
    void methodC();
}

abstract class SubClassA implements SuperInterfaceC{
    @Override
    public void methodA() {
        //
    }
}

class SubClassB extends SubClassA{

    @Override
    public void methodC() {
        //
    }
    @Override
    public void methodB() {
        //
    }
}
```

인터페이스

- 인터페이스에서 선언된 멤버 데이터는 public final static 지정자가 생략된 생태이다.
- 인터페이스는 생성자가 없기 때문에 항상 자식 클래스로 생성해야 한다.

```
interface School{
    String SCHOOL_NAME="SSU";
}

public class InterfaceTest {
    public static void main(String[] args) {
            System.out.println(School.SCHOOL_NAME);
    }
}
```

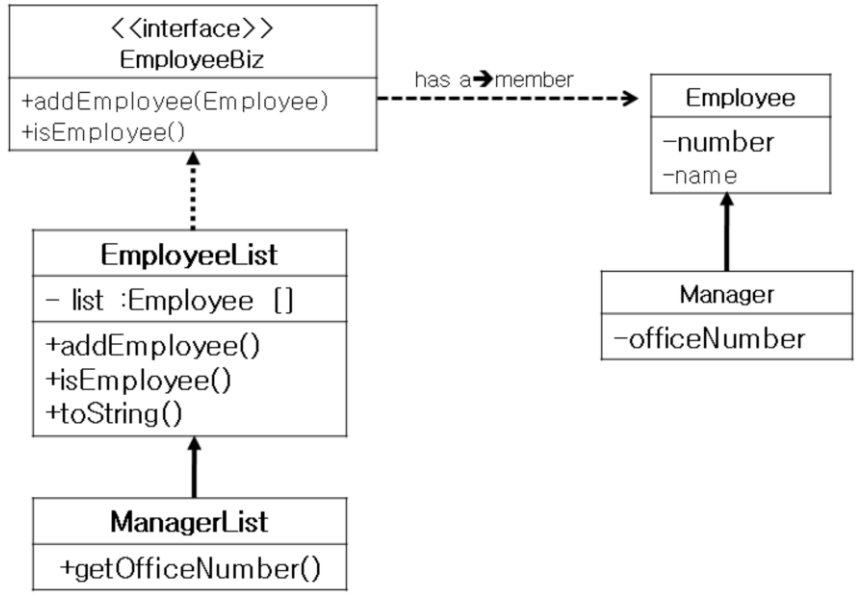
9-16

class, abstract class, interface

	생성자	자식 클래스	다중 상속	내용
interface	없다	필수	O	메서드 중심의 비즈니스 설계 데이터는 모두 public final static
abstract class	new 불가	필수	X	정의된 메서드중 공통 구현부 정의 this(), super()로만 생성자 사용
class	사용 가능	선택	X	완벽한 객체

9-17

class, abstract class, interface



9-18

F11 : 디버깅 시작
F8 : 디버깅 계속
F4 : 이클립스 클래스 구조 확인
Ctrl + T : 이클립스 클래스 계층 구조 팝업
Ctrl + Shift + C : 이클립스 한줄 주석
Ctrl + Shift + T : 블록 객체 찾기