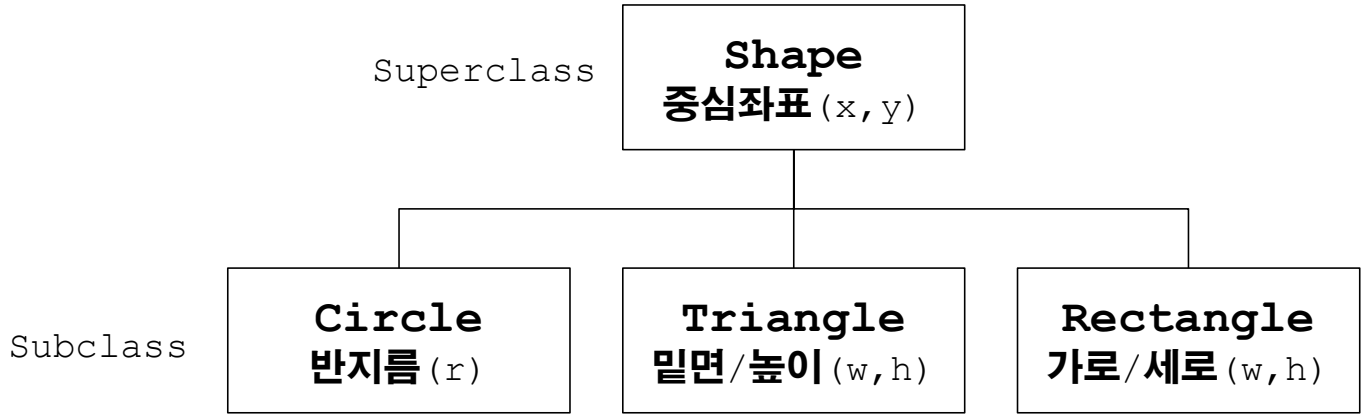
03-5. 상속과 다형성

Java 객체지향

상속(Inheritance)

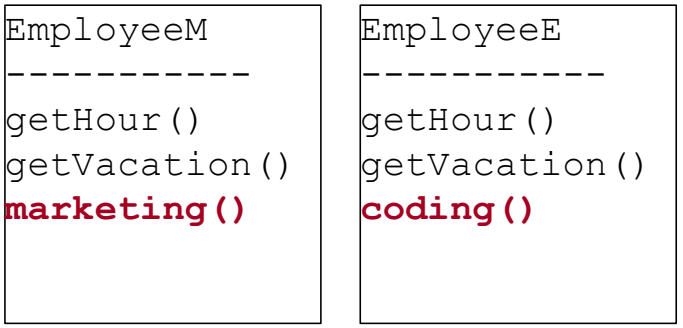
- 상속(Inheritance)의 개념
  - **상속이란 부모 클래스(parent class)의 멤버 변수와 멤버 메쏘드를 물려 받는 것을 의미**
  - **자식 클래스(child class)는 부모 클래스의 모든 속성을 물려받음.**
  - **자식 클래스는 부모로부터 상속받은 속성 외에 자신만의 속성을 지닐 수 있음**



상속(Inheritance) – 개념(직원 규정집의 예)

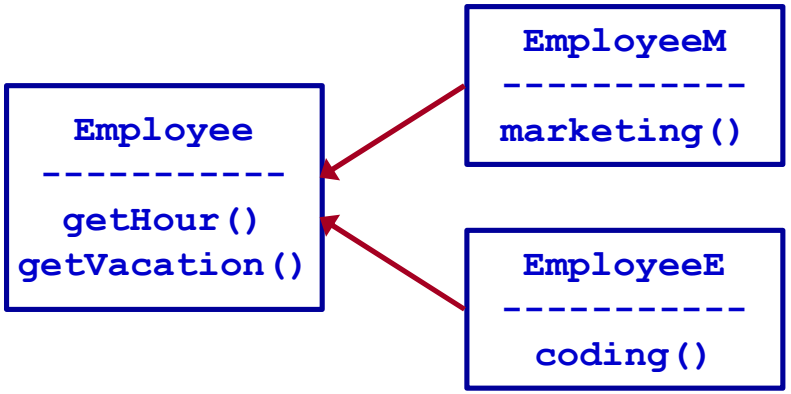
- 직군별 규정집을 별도로 관리할 경우
  - **G 직군 규정집 : 30 페이지**
  - **E 직군 규정집 : 25 페이지**
  - **M 직군 규정집 : 22페이지**
  - **공통규정이 바뀌었을 경우 세 벌의 규정집 수정 필요.**
  - **특정 직군에만 해당되는 규정을 알아내기 어려움**

- 직원 공용 규정
  - **주당 40시간 근무**
  - **매년 2주의 휴가**
- 직군별 규정
  - **E 직군 직원은 coding()이 가능함**
  - **M 직군 직원은 marketing()이 가능함**



상속(Inheritance) – 개념(직원 규정집의 예)

- 공용 규정집을 두고 직군별로 상이한 규정만 따로 관리할 경우
  - **공용 규정집 : 20 페이지**
  - **G 직군 규정집 : 공용 + 10 페이지**
  - **E 직군 규정집 : 공용 + 5 페이지**
  - **M 직군 규정집 : 공용 + 2 페이지**
  - **공통 규정이 바뀌었을 경우 공용 규정집만 수정하면 됨.**
  - **특정 직군별 규정을 알기 쉬움**



❖ 장점

- 유지관리 용이성(maintenance)
  - 공용 규정이 변화가 있을 경우, Employee 클래스의 규정만 수정해줌
- 구역성(locality)
  - E직군, M직군 관련 전체 규정들을 쉽게 찾아볼 수 있음
- 재사용성 (code reusability)
  - 한차례 프로그램을 작성한 후, 여러 곳에서 가져다 쓸 수 있음

상속(Inheritance)

- 상속(Inheritance)의 특성
  - **상속되는 것**
    - **대부분의 멤버 변수와 멤버 메쏘드**
  - **상속되지 않는 것**
    - **명시적으로 숨긴 변수 (Private Access Limitation)**
    - **생성자 (생성자는 멤버 메쏘드가 아님)**
    - **상위클래스에 존재하는 멤버변수와 동일한 이름의 멤버변수를 하위클래스에 선언한 경우**
  - **JAVA에서는 다중 상속을 허용하지 않음 → 인터페이스를 이용하여 유사한 효과를 얻음**

상속(Inheritance) – 예제

```
public class Shape {
   int x; int y;
   public Shape(int x, int y) {
          this.x = x; this.y = y;
   }
}
```

```
public class Circle extends Shape{
   int r;
   public Circle() {
          this.r = 1;
   }
}
public class ShapeTest {
   public static void main(String[] args) {
          Shape s = new Shape(50,30);
          Circle c = new Circle(); //오류 발생
   }
}
```

상속(Inheritance)

- super 키워드
  - **상속된 객체의 생성시 부모 클래스의 생성자가 먼저 호출되고 자식 클래스의 생성자가 호출 됨.**
  - **명시적으로 부모 클래스의 생성자를 호출하지 않을 경우 부모 클래스의 기본 생성자가 수행됨.**
  - **Shape 클래스에는 기본 생성자인 Shape()가 존재하지 않으므로 Circle() 생성자에서 암묵적으로 Shape() 를 호출할 수 없기 때문에 오류 발생**
  - **상위 클래스의 생성자 super() 이용하여 해결 가능**

```
public class Circle extends Shape {
   int r;
   public Circle() {
          super(50,30);
          this.r = 1;
   }
}
```

접근제한

- public
  - 모든 접근을 허용
  - 대부분의 생성자(Constructor)와 타인과 공유할 클래스와 메쏘드에 사용함 (예: Getters / Setters)
- private
  - 현재의 객체 내에서만 접근 허용
  - 멤버 변수와 타인과 공유하지 않을 메쏘드에 사용함
- protected
  - 같은 패키지에 있는 객체와 상속 관계의 객체들만 허용
- default (Package) : 아무런 접근제한을 표시하지 않음
  - 같은 패키지 내에 있는 객체들만 허용

구분	public	protected	package	private
해당 클래스	O	O	O	O
서브 클래스	O	O	X	X
패키지 클래스	O	O	O	X
모든 클래스	O	X	X	X

오버라이딩(Overriding)

- 오버라이딩(Overriding)
  - 메쏘드 다중 정의
  - 동일한 메쏘드 이름과 동일한 시그너쳐를 가짐
  - 한 메쏘드는 부모 클래스에 다른 메쏘드는 자식 클래스에 존재

- 오버로딩(Overloading)
  - 메쏘드 재정의
  - 동일한 메쏘드 이름에 상이한 시그너쳐를 가짐
  - 두 메쏘드 모두 한 클래스 내에 선언 됨

오버라이딩(Overriding)

오버라이딩(Overriding)

```
class Shape {
  public void getArea() {

        System.out.println("Shape");
  }
}

class Circle extends Shape {
  public void getArea() {

  System.out.println("Circle");
  }
}
```

오버로딩(Overloading)

```
class Circle extends Shape {
  public void getArea() {

        System.out.println("0");
  }

  //오버로딩된 메쏘드
  public void getArea(int r) {

        System.out.println(r);
  }

}
```

final

- 변수를 선언할 때 final 키워드를 사용하면 초기화된 변숫값을 수정할 수 없도록 제한합니다.
- final 변수를 사용하는 목적은 다음과 같습니다.
  - 초깃값을 변경하지 않고 그대로 사용하기 위해
  - 파이(원주율)와 같은 수학적 값을 사용하기 위해

❑ 메서드를 선언할 때 final 키워드를 사용하면 메서드를 오버라이딩할 수 없도록 제한합니다.

❑ 클래스를 선언할 때 final 키워드를 사용하면 클래스를 상속할 수 없도록 제한합니다.

객체 자료유형 변환

- 올림변환(Upcasting)
  - **부모 클래스의 레퍼런스에 자식 클래스의 인스턴스 할당 가능 (자동)**
- 내림변환(Downcasting)
  - **자식 클래스의 레퍼런스에 부모 클래스의 객체를 할당하려면 명시적으로 변환 필요**

```
Shape s1 = new Circle(); //올림변환
Circle s2 = new Circle();
Shape s3 = new Shape(0,0);

s1.draw(); //컴파일 오류 - draw() 메쏘드는 Circle 클래스에만 정의됨
(Circle)s1.draw(); //가능
s2.draw(); //가능
s2 = s3; //컴파일 오류 - 명시적 변환 필요
s2 = (Circle)s3; //컴파일 성공, 런타임 오류
```

객체 자료유형 변환

- instanceof 연산자
  - **객체의 변환에서 변환을 잘못 수행하면 심각한 오류 발생**
  - **사전에 변환이 가능한지 조사할 수 있는 연산자**
  - **"객체" instanceof "클래스"**
  - **객체와 조사할 클래스가 상관관계를 갖지 않으면 컴파일 오류 발생**

```
Shape s1 = new Circle(); //올림변환
Circle s2 = new Circle();
Shape s3 = new Shape(0,0);

if( s1 instanceof Circle ) {} // true
if( s2 instanceof Shape ) {} // true
if( s3 instanceof Circle ) {} // false
```

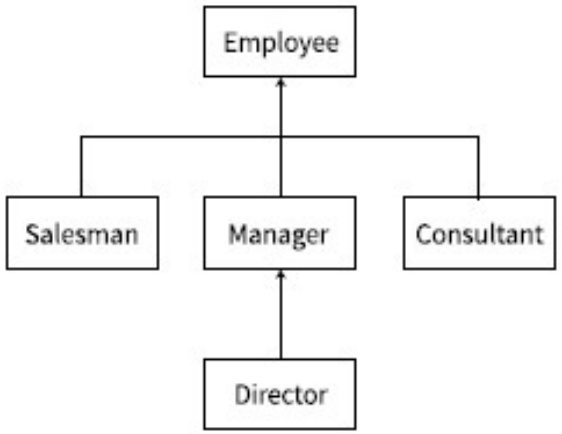
다형성(Polymorphism)

- 바인딩(Binding)
  - **실행할 메쏘드의 주소(객체)를 결정하는 것**
  - **정적바인딩 – 컴파일시 결정**
  - **동적바인딩 – 실행시간에 결정**

- 다형성(Polymorphism)
  - **서로 다른 객체가 같은 메쏘드에 대하여 다른 행위를 하도록 동적 바인딩을 사용하는 것**
  - **상속된 객체에 대해 부모 클래스의 메쏘드를 호출 하였을 때 그 메쏘드에서 사용되는 다른 메쏘드가 자식 클래스에 다시 정의 (override) 된 경우 자식 클래스의 메쏘드가 동적으로 바인딩 됨**

다형성(Polymorphism)

- 정적 바인딩




```
Salesman s = new Salesman();
Consultant c = new Consultant();
Director d = new Director();
```

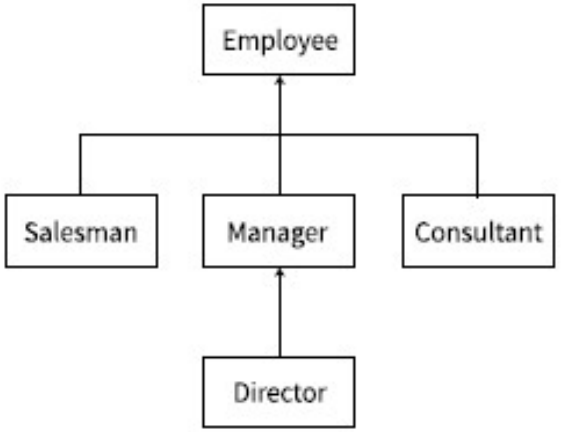
```
// Salesman 소득세 계산
calcTax(s);
// Consultant 소득세 계산
calcTax(c);
// Director 소득세 계산
calcTax(d);
```

```
public static void calcTax(Salesman s) {
  // 인자로 전달된 인스턴스의 소득세 계산
}
public static void calcTax(Consultant c) {
  // 인자로 전달된 인스턴스의 소득세 계산
}
public static void calcTax(Director d) {
  // 인자로 전달된 인스턴스의 소득세 계산
}
```

다형성(Polymorphism)

- 다형성 적용

```
public static void calcTax(Employee e) {
  // 인자로 전달된 인스턴스의 소득세 계산
}
```




```
public static void main(String[ ] args) {
    Salesman s = new Salesman( );
    Consultant c = new Consultant( );
    Director d = new Director( );

    calcTax(s);
    calcTax(c);
    calcTax(d);
}

public static void calcTax(Employee e) {
    // 인자로 전달된 인스턴스의 소득세 계산

}
```

다형성(Polymorphism)

HRSTest.java

```
package com.ruby.java.ch08.polymorphism;
...
public class HRSTest {
  public static void calcTax(Employee e) {
    System.out.println("소득세를 계산합니다.");
  }

  public static void main(String[] args) {
    Salesman s = new Salesman();
    Consultant c = new Consultant();
    Director d = new Director();

    calcTax(s);
    calcTax(c);
    calcTax(d);
  }
}
```

감사합니다

단단히 마음먹고 떠난 사람은
산꼭대기에 도착할 수 있다.
산은 올라가는 사람에게만 정복된다.

윌리엄 셰익스피어
William Shakespeare