- 12. 패턴
- (3) 스트레티지 패턴 (Strategy Pattern)

객체가 가지는 각각의 기능을 부품화 하여 각각의 클래스나 인터페이스에 담아두고, 필요할 때 언제든지 메인 클래스에 적용할 수 있도록 만든 패턴

- ※ 세가지 유형의 로봇의 각각의 특징을 통해 보는 스트레티지 패턴을 작성하는 방법
- 1) 각각의 클래스 정의
- 2) 공통점을 Super Class로 올려서 추상클래스로 만든다.
- 3) 각 기능을 부품화하여 각각의 인터페이스 및 클래스에 정의한다. (object modulation) <ex1> 그림으로 정리하면 다음과 같다.

Robot(추상) actionWalk(){걸을수있다} actionRun(){뛸수있다} Shape(){팔다리몸} actionFly();actionMissile();actionKnife();(추상) SuperRobot StandardRobot LowRobot Shape(){} Shape(){} Shape(){} actionFly(){X} actionFly(){X} actionFly(){O} actionMissile(){O} actionMissile(){0} actionMissile(){X} actionKnife(){L} actionKnife(){W} actionKnife(){X}

- Robot: Super 클래스이자 추상 클래스로 세 로봇들의 공통점이 모여있다.
- 파란글씨로 쓰여진 각각의 로봇의 특징들은 하나씩 클래스로 들어가서 부품화된다.

```
<ex1-1> - 부품화된 로봇의 특징 중 한가지의 예: 로봇이 날 수 있는지 여부를 인터페이스로<br/>public interface FlyImpl {<br/>public void fly(); }<ex1-2> - '날수 있는 로봇'의 특징을 부품화 하여, 클래스에 ex1-1의 인터페이스의 추상메소드를 오버라이드<br/>public class FlyYes implements FlyImpl {<br/>@Override<br/>public void fly() {<br/>System.out.println("날 수 있습니다.");
```