1. FAÇADE 패턴
   1. 개념 : 우리가 잘~아는 객체지향 프로그래밍에서 클래스별로 기능 분할해서 개발하는것
2. 템플릿-메소드 패턴
   1. 개념 : 작업의 세부 방식을 다양화하는데 사용하는 패턴  
      EX) 약과를 만들 때, 1) 반죽 2) 튀김 3) 시럽시럽 이 세 과정을 어떻게 하느냐에 따라 수많은 약과들이 탄생한다.  
      전략 패턴이 이 각각의 기능을 모듈화해서 갈아 끼워넣는다면, 템플릿-메소드 패턴은 각각의 기능을 자식 클래스에서 오버라이딩해서 구현하는 것이다.  
      (전반적인 과정은 부모 클래스 수행-1)2)3), 세부 메소드를 자식 클래스에서 호출)
   2. 작성법
      1. 공통으로 움직일 큰 기능을 부모 클래스에서 선언한다.
      2. 각 메소드별 세부 행위를 오버라이딩받은 자식 클래스에서 세분화해서 구현한다.
3. 데코레이터 패턴
   1. 개념 : 특정 클래스의 객체들이 할 수 있는 일을 여러가지 두고, 각 객체마다 사용자가 원하는대로 골라 시키거나 이런저런 기능을 장착할 때 사용한다.  
      EX) 1945 비행기 아이템 먹을때마다 공격력 업그레이드
   2. 작성법
      1. 추상 클래스로 데코레이터를 만든 뒤, 일할 객체를 생성자로 받아온다.
      2. 그 다음으로 위 객체의 행위(메소드를) 오버라이딩해서 가져온다.
      3. 추상클래스를 상속받은 자식 클래스(기능별 행위)에서 ‘super.부모클래스의 메소드’ 명령어를 통해 부모 방식 그대로 행위를 한 뒤, 자식별로 추가 기능들을 수행한다.
4. 팩토리-메소드 패턴
   1. 개념 : 객체를 생성자를 통해 직접 불러오는 것이 아니라, 팩토리클래스에서 한번 거쳐서 가져오는 것?  
      장점 1 – 생성자 대상이 되는 클래스를 바꿀 때, 이를 생성자로 활용한 코드들을 전부 고친 형식에 맞게 수정할 필요가 없다.  
      장점 2 – 복잡하고 어려운 코드들을 전부 알 필요 없이, 팩토리 메소드를 통해 거쳐서 간편하게 가져올 수 있다.  
      장점 3 – 무지 다양한 생성자들별로 구분해서 정리가 필요할 경우(e-버튼, 콤보 등 컴포넌트가 라이트ver 다크ver 섞여있는 경우), 팩토리 메소드를 통해 엮어서 한번에 편하게 가져올 수 있다.
5. 추상 팩토리-메소드 패턴
   1. 개념 : 위 팩토리 메소드 패턴에, 추상화가 한단계 더 입혀진 것  
      => 팩토리-메소드를 추상클래스를 통해 한번(또는 여러 번) 또 거치는 것
6. Mediator(미디에이터)(중재자) 패턴
   1. 개념 : 특정 클래스에서 이벤트가 발생할 때 마다 연결된 다른 클래스들에게 이를 알려야 하는 경우 사용하는 디자인 패턴  
      ex) 거래처목록 화면에서 리스트보기 / 갤러리보기 모드를 누르면, 반대쪽 모드를 숨기고 선택한 모드를 보인 뒤, 새 모드에 맞게 화면에 대한 다운로드 명령을 다시 내려야한다.  
      위 명령들을 다 생성자로 호출하고…코드에 넣고 하면 얼마나 복잡하겠어?  
      => 미디에이터(중재자)를 통해 위 항목들을 중간 단계를 거쳐 작업 rr
   2. 리스트뷰와 갤러리뷰 관련 클래스(코드)들이 공통적으로 모드에 반응하는 인터페이스를 장착하고(부모 인터페이스를 만들어서 클래스들에 상속하고),  
      => 거래처 목록 리스트화면에 위 인터페이스를 적용
   3. 즉, 여러 클래스들의 관계가 특정 이벤트를 중심으로 복잡하게 얽힌 설계에서 유용하게 사용 가능하다~
7. 컴포지트 패턴
   1. 개념 : 객체 간 트리구조(전체-부분)를 통해 관계를 정의하며, 복합객체(폴더)와 단일객체(파일)를 구별없이 다루게 하는 패턴.  
      ex) 컴퓨터 폴더 / 파일?