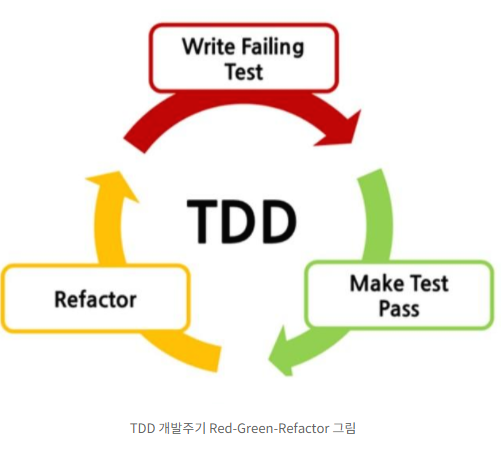
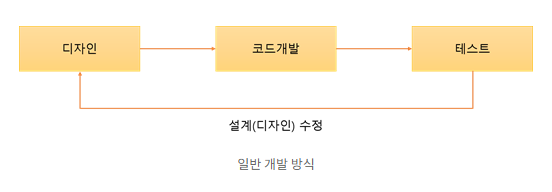
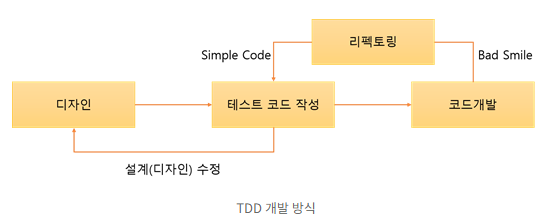
**TDD(Test-Driven-Development)**

1. **TDD**- 테스트 주도 개발, 반복테스트를 이용한 소프트웨어 방법론  
   - 작은 단위 테스트 케이스를 작성, 이를 통과하는 코드를 추가하는 단계를 반복해 구현한다.  
   - TDD의 개발주기  
   <RED> : 실패하는 테스트 코드를 먼저 작성한다.  
   <Green> : 테스트 코드를 성공시키기 위한 실제 코드를 작성한다.  
   <Yellow> : 중복 코드 제거, 일반화 등의 리팩토링을 수행한다.  
   실패하는 테스트 코드를 작성할 때까지 실제 코드를 작성하지 않는 것과, 실패하는 테스트를 통과할 정도의 최소 실제 코드를 작성해야 한다. 이를 통해 실제 코드에 대해 기대되는 바를 보다 명확하게 정의함으로써 불필요한 설계를 피할 수 있고, 정확한 요구 사항에 집중할 수 있다.
2. **일반 개발 방식과 TDD 개발 방식의 비교**- 일반 개발 방식  
     
   : ‘요구사항 분석 -> 설계 -> 개발 -> 테스트 -> 배포’의 형태의 개발 주기를 갖는다.  
   : 이러한 방식은 소프트웨어 개발을 느리게 하는 잠재적 위험이 존재한다.  
   1) 소비자의 요구사항이 처음부터 명확하지 않을 수 있다.  
   2) 따라서 처음부터 완벽한 설계는 어렵다.  
   3) 자체 버그 검출 능력 저하 또는 소스 코드의 품질이 저하될 수 있다.  
   4) 자체 테스트 비용이 증가할 수 있다.  
   : 초기설계가 완벽하지 않고, 완벽을 위한 재설계로 인해 코드를 삽입, 수정, 삭제하는 과정에서 불필요한코드가 남거나 중복처리 될 가능성이 크다. 즉, 이러한 코드들은 재사용이 어렵고 관리가 어려워서 유지보수를 어렵게 만든다.  
   : 작은 부분의 기능 수정에도 모든 부분을 테스트를 하므로 전체적인 버그를 검출하기 어렵다.  
   - TDD 개발 방식  
     
   : 가장 큰 차이점은 테스트 코드를 작성한 뒤에 실제 코드를 작성한다는 점.  
   : 디자인 단계에서 프로그래밍 목적을 반드시 미리 정의해야만 하고, 또 무엇을 테스트해야 할지 미리 정의(테스트 케이스 작성)해야만 한다.  
   : 테스트코드를 작성하는 도중에 발생한 예외 사항(버그, 수정사항)들은 테스트케이스에 추가하고 설계를 개선한다. 이후 테스트가 통과된 코드만을 코드 개발 단계에서 실제코드로 작성.  
   : 이러한 반복적인 단계가 진행되면서 코드의 버그가 줄어들고, 소스코드는 간결해진다.
3. **TDD 개발 방식의 장단점**- 장점  
   1) 보다 튼튼한 객체 지향적인 코드 생산  
   2) 재설계 시간의 단축  
   3) 디버깅 시간의 단축  
   4) 테스트 문서의 대체 기능  
   5) 추가 구현의 용이함  
   - 단점  
   1) 생산성의 저하  
   2) 개발 시간의 증가