제어 플래그 삭제 (Remove Control Flag)

제어 플래그 사용이 많아 코드가 복잡해지는 경우, 제어 플래그를 삭제하고 break, continue, return을 이용해서 단순화한다.

데이터 파일

.\src\dbfile.txt

```
aaa@test.com=aaaaa
gdhong@text.com=Hong Gil-Dong
crpark@test.com=Park Changryum
```

메인

데이터 파일의 경로를 SimpleDatabase 생성자의 파라미터로 전달
SimpleDatabase 인스턴스를 통해 HashMap에 저장되어 있는 메일과 이름을 출력

```
import java.io.FileReader;
import java.util.Iterator;

public class SimpleDatabaseMain {
	public static void main(String[] args) {
		String filePath = "./src/dbfile.txt";
		SimpleDatabase db;
		try {
			db = new SimpleDatabase(new FileReader(filePath));
			Iterator<String> it = db.iterator();
			while (it.hasNext()) {
				String key = it.next();
				System.out.println("KEY: " + key);
				System.out.println("VALUE: " + db.getValue(key));
				System.out.println();
			}
		} catch (Exception e) {
			e.printStackTrace();
		}

	}
}
```

단순 구현

문제점 1. 의미를 파악할 수 없는 변수 이름을 사용
문제점 2. 제어 플래그를 많이 사용하고 있어 로직을 파악하기 어려움

```
public class SimpleDatabase {
	private Map<String, String> map = new HashMap<String, String>();
```

```
	public SimpleDatabase(Reader r1) throws IOException {
		BufferedReader r2 = new BufferedReader(r1);
		boolean flag = false;
		String tmp;
		while (!flag) {
			tmp = r2.readLine();
			if (tmp == null) {
				flag = true;
			} else {
				boolean flag2 = true;
				StringBuffer s1 = new StringBuffer();
				StringBuffer s2 = new StringBuffer();
				for (int i = 0; i < tmp.length(); i++) {
					char tmp2 = tmp.charAt(i);
					if (flag2) {
						if (tmp2 == '=') {
							flag2 = false;
						} else {
							s1.append(tmp2);
						}
					} else {
						s2.append(tmp2);
					}
				}
				String ss1 = s1.toString();
				String ss2 = s2.toString();
				map.put(ss1, ss2);
			}
		}
	}

	public String getValue(String key) {
		return map.get(key);
	}

	public Iterator<String> iterator() {
		return map.keySet().iterator();
	}
}
```

개선1. 의미있는 변수명으로 변경 (Ctrl+Shift+R)

```
import java.io.BufferedReader;
import java.io.IOException;
import java.io.Reader;
import java.util.HashMap;
import java.util.Iterator;
import java.util.Map;

public class SimpleDatabase {
	private Map<String, String> map = new HashMap<String, String>();
```

```
	public SimpleDatabase(Reader reader) throws IOException {
		BufferedReader bufferedReader = new BufferedReader(reader);
		boolean reading = false;
		String line;
		while (!reading) {
			line = bufferedReader.readLine();
			if (line == null) {
				reading = true;
			} else {
				boolean scanningKey = true;
				StringBuffer keyBuffer = new StringBuffer();
				StringBuffer valueBuffer = new StringBuffer();
				for (int i = 0; i < line.length(); i++) {
					char c = line.charAt(i);
					if (scanningKey) {
						if (c == '=') {
							scanningKey = false;
						} else {
							keyBuffer.append(c);
						}
					} else {
						valueBuffer.append(c);
					}
				}
				String key = keyBuffer.toString();
				String value = valueBuffer.toString();
				map.put(key, value);
			}
		}
	}

	public Iterator<String> iterator() {
		return map.keySet().iterator();
	}

	public String getValue(String key) {
		return map.get(key);
	}
}
```

개선2. 불필요한 제어 플래그를 삭제

```
import java.io.BufferedReader;
import java.io.IOException;
import java.io.Reader;
import java.util.HashMap;
import java.util.Iterator;
import java.util.Map;

public class SimpleDatabase {
	private Map<String, String> map = new HashMap<String, String>();

	public SimpleDatabase(Reader reader) throws IOException {
```

```
            BufferedReader bufferedReader = new BufferedReader(reader);
            boolean reading = false;   ⇐ 삭제
            String line;
            while (true) {
                  line = bufferedReader.readLine();
                  if (line == null) {
                        break;
                  }
                  else {               ⇐ 삭제

                  boolean scanningKey = true;
                  StringBuffer keyBuffer = new StringBuffer();
                  StringBuffer valueBuffer = new StringBuffer();
                  for (int i = 0; i < line.length(); i++) {
                        char c = line.charAt(i);
                        if (scanningKey) {
                              if (c == '=') {
                                    scanningKey = false;
                              } else {
                                    keyBuffer.append(c);
                              }
                        } else {
                              valueBuffer.append(c);
                        }
                  }
                  String key = keyBuffer.toString();
                  String value = valueBuffer.toString();
                  map.put(key, value);

                  }                    ⇐ 삭제
            }

      }

      public Iterator<String> iterator() {
            return map.keySet().iterator();
      }

      public String getValue(String key) {
            return map.get(key);
      }
}
```

개선3. indexOf() 메소드를 이용하여 단순화

```
import java.io.BufferedReader;
import java.io.IOException;
import java.io.Reader;
import java.util.HashMap;
import java.util.Iterator;
import java.util.Map;

public class SimpleDatabase {
```

```
	private Map<String, String> map = new HashMap<String, String>();

	public SimpleDatabase(Reader reader) throws IOException {
		BufferedReader bufferedReader = new BufferedReader(reader);
		String line;
		while (true) {
			line = bufferedReader.readLine();
			if (line == null) {
				break;
			}

			int equalIndex = line.indexOf("=");
			if (equalIndex > 0) {
				String key = line.substring(0, equalIndex);
				String value = line.substring(equalIndex + 1);
				map.put(key, value);
			}
		}
	}

	public Iterator<String> iterator() {
		return map.keySet().iterator();
	}

	public String getValue(String key) {
		return map.get(key);
	}
}
```

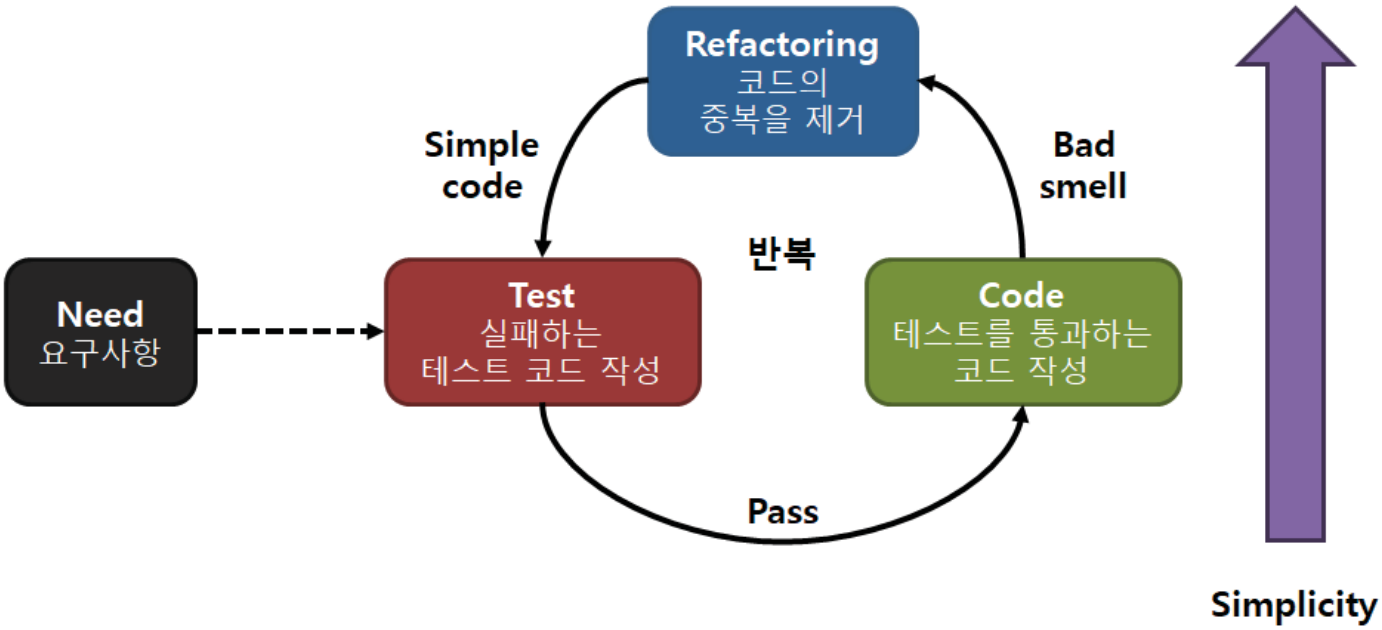
개선4. 정규식을 이용하여 단순화

```
import java.io.BufferedReader;
import java.io.IOException;
import java.io.Reader;
import java.util.HashMap;
import java.util.Iterator;
import java.util.Map;
import java.util.regex.Matcher;
import java.util.regex.Pattern;

public class SimpleDatabase {
	private Map<String, String> map = new HashMap<String, String>();

	/**************************************************************************
		(		group(1)로 캡처할 문자열 시작
			[^=]+	'='를 제외한 문자로 구성된 문자 클래스 1개 이상 반복
		)		group(1) 끝
		=		'=' 문자
		(		group(2)로 캡처할 문자열 시작
			.*	임의의 문자 0개 이상 반복
		)		group(2) 끝
	**************************************************************************/
	private static Pattern dataPattern = Pattern.compile("([^=]+)=(.*)");
```

```
	public SimpleDatabase(Reader reader) throws IOException {
		BufferedReader bufferedReader = new BufferedReader(reader);
		String line;
		while (true) {
			line = bufferedReader.readLine();
			if (line == null)
				break;

			Matcher matcher = dataPattern.matcher(line);
			if (matcher.matches()) {
				String key = matcher.group(1);
				String value = matcher.group(2);
				map.put(key, value);
			}
		}
	}

	public Iterator<String> iterator() {
		return map.keySet().iterator();
	}

	public String getValue(String key) {
		return map.get(key);
	}
}
```

TDD(Test Driven Development)

- **Clean code that works!**



xUnit, JUnit

- **xUnit**
  - 언어별 다양한 단위 테스트 도구가 존재
  - JUnit, NUnit, CPPUnit, PyUnit, VBUnit, PearlUnit 등

- **JUnit**
  - 1997년 에릭 감마와 켄트 백이 제작
  - Java를 위한 단위 테스트 프레임워크
  - 문자 혹은 GUI 기반으로 실행
  - 단정문으로 테스트 케이스의 수행 결과를 판별
  - 어노테이션으로 간결하게 지원
  - 결과를 성공(녹색), 실패(붉은색) 중 하나로 표시

JUnit Annotation

https://junit.org/junit5/docs/current/user-guide/#writing-tests-annotations

어노테이션	설명
`@Test`	실행할 테스트 메소드 앞에 붙임
`@Test(expected)`	발생할 것으로 예상되는 예외를 지정 예외가 발생하지 않으면 실패
`@Test(timedout)`	테스트가 끝나는 시간을 예측 지정된 시간 보다 길게 테스트가 진행되면 실패
`@Ignore`	다음에 오는 테스트를 무시 테스트를 하지 않을 메소드 앞에 붙임
`@Before, @After`	각 단위 테스트 메소드의 실행 전후에 초기화와 자원정리 작업을 수행
`@BeforeClass, @AfterClass`	각 단위 테스트 클래스의 실행 전후에 초기화와 자원정리 작업을 수행
`@RunWith`	지정된 러너가 아닌 사용자가 지정한 러너를 통해 특정 클래스를 실행
`@SuiteClasses`	테스트하려고 하는 여러 클래스를 지정
`@Parameters`	여러 개의 파라미터 값을 테스트하려고 할 때 자동적으로 테스트를 실행

JUnit Method

http://junit.sourceforge.net/javadoc/org/junit/Assert.html

메소드	설명
`assertEquals(A, B)`	A와 B가 일치하는지를 조사한다. A나 B에는 Object, int, float, long, char, boolean, ... 등의 모든 자료형이 들어갈 수 있다. 단 A, B의 타입은 언제나 같아야만 한다.
`assertTrue(X)`	X가 참인지를 조사한다. X는 boolean 형태의 값이어야 한다.
`assertFalse(X)`	X가 거짓인지를 조사한다. X는 boolean 형태의 값이어야 한다.
`fail(message)`	테스트가 위 문장을 만나면 message를 출력하고 무조건 실패하도록 한다.
`assertNotNull(Object X)`	X가 Null이 아닌지를 조사한다. 만약 Null이라면 assertionFailedError가 발생한다.
`assertNull(Object X)`	X가 Null인지를 조사한다. 만약 Null이 아니라면 assertionFailedError가 발생한다.
`assertSame(Object A, Object B)`	A와 B가 같은 객체인지를 조사한다.

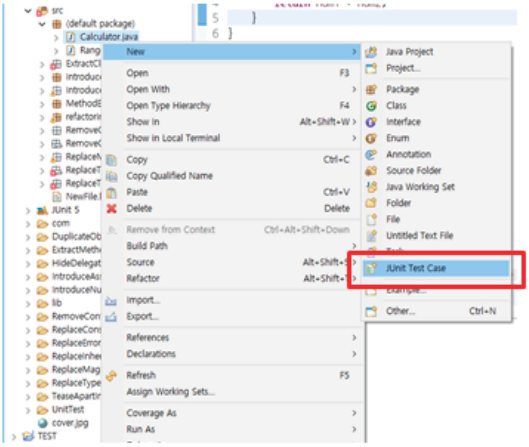
JUnit 실습

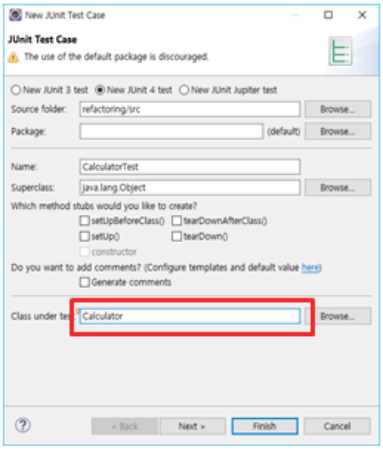
▪ 프로덕트 코드

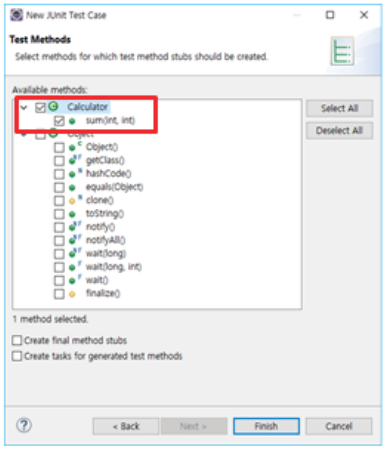
```
public class Calculator {
    public int sum(int num1, int num2) {
        return num1 + num2;
    }
}
```

▪ 테스트 케이스 생성

- New > JUnit Test Case



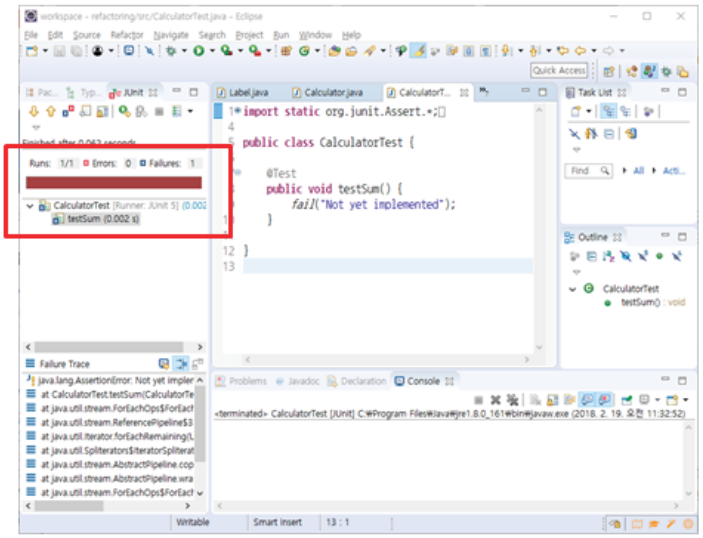




JUnit 실습

- **단위 테스트 실행**

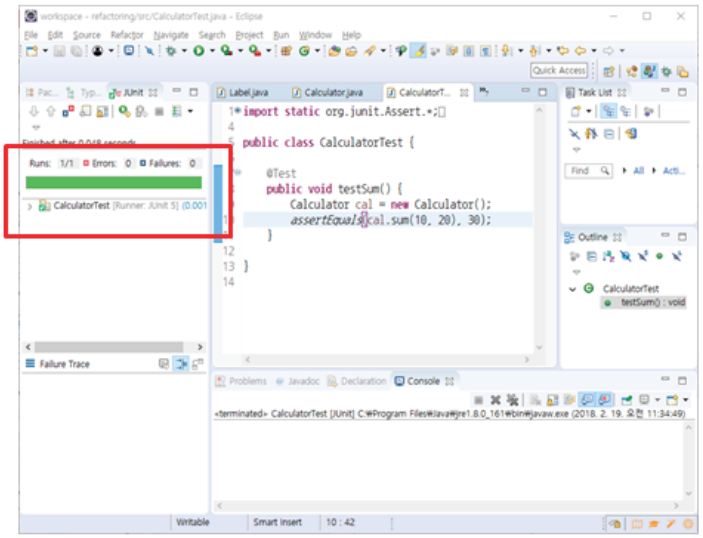
```
import static org.junit.Assert.*;
import org.junit.Test;

public class CalculatorTest {

  @Test
  public void testSum() {
      fail("Not yet implemented");
  }

}
```



JUnit 실습

- **단정문을 이용한 단위 테스트**

```
import static org.junit.Assert.*;
import org.junit.Test;

public class CalculatorTest {

  @Test
  public void testSum() {
      Calculator cal = new Calculator();
      assertEquals(cal.sum(10, 20), 30);
  }

}
```



JUnit 실습

- **어노테이션을 이용한 단위 테스트**

```
import static org.junit.Assert.*;
import org.junit.Test;

public class CalculatorTest {

  @Test(timeout=3000)
  public void testSum() {
      Calculator cal = new Calculator();
      assertEquals(cal.sum(10, 20), 30);
  }

}



public class Calculator {
  public int sum(int num1, int num2) {
      try {
          Thread.sleep(4000);
      } catch (Exception e) { }
      return num1 + num2;
  }
}
```