Chapter 17

Appendix 11. Java Reserved keywords

List of Reserved Java Keywords

abstract	assert	boolean	break	byte	case
catch	char	class	const	continue	default
double	do	else	enum	extends	false
final	finally	float	for	goto	if
implements	import	instanceof	int	interface	long
native	new	null	package	private	protected
public	return	short	static	strictfp	super
switch	synchronized	this	throw	throws	transient
true	try	void	volatile	while	

*The *strictfp* keyword was added to this list in Java Standard Edition version 1.2, *assert* in version 1.4, and *enum* in version 5.0.

Even though *goto* and *const* are no longer used in the Java programming language, they still cannot be used as keywords.

- 1) assert
- 2) volatile
- 3) strictfp
- 4) transient

1) assert

Assertion Checks (#4-2, page 281)

- Assertion : something that says something about the state of the program
  - » Can be true or false
  - » Should be true when no mistakes in running program

Assertion Checks

- Example found in comments

```
//n == 1
  while (n < limit)
  {
  n = 2 * n;
  }
//n >= limit
//n is the smallest power of 2 >= limit
```

- Syntax for assertion check

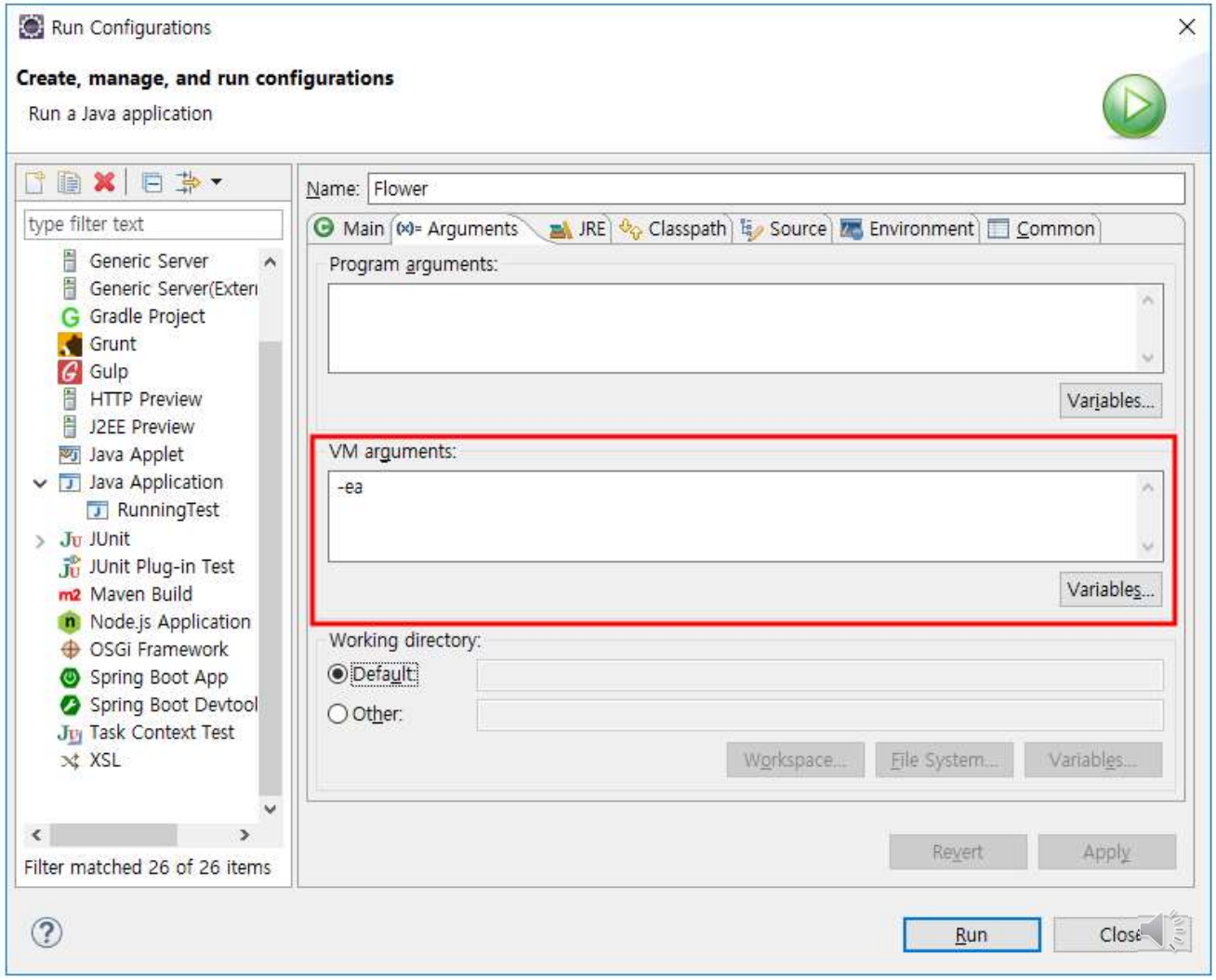
```
Assert Boolean_Expression;
```

Assertion Checks

- Equivalent example using **`assert`**

```
assert n == 1;
while (n < limit)
{
n = 2 * n;
}
assert n >= limit;
//n is the smallest power of 2 >= limit.
```


- 이클립스에서 assert가 동작
  - » Run Configurations 에서 VM arguments에 -ea 를 추가해서 실행



- 사용법

```
assert expression1;
assert expression1: expression2;
```

» 1) 인자로 boolean으로 평가되는 표현식 또는 값을 받아서 참이면 그냥 지나가고, 거짓이면 AssertionError 예외가 발생

» 2) 표현식1이 거짓인 경우 두번째 표현식의 값이 예외 메세지로 보여짐

»

```java
package asserttest;
public class Counter {
	private int count;
	public Counter(final int initialCount) {

	  //assert initialCount >= 0;
	  if(initialCount < 0) {
	    throw new IllegalArgumentException("초기값은 0이상이어야 합니다: " + initialCount);
	  }
	  this.count = initialCount;

	}


	private void changeCount(final int val) {
	  // 사후 조건 체크용
	  final int preCount = this.count;

	  // 사전 조건 체크
	  assert val == 1 || val == -1;

	  if((this.count + val) < 0) {
	    this.count = 0;
	  } else {
	    this.count += val;
	  }

	 // count = 0; //assertTest#3
	  // 사후 조건 체크
	  assert (val == 1 && (this.count > preCount)) || (val == -1 && (this.count <= preCount));

	 // count = -1 ; //assertTest#4
	  //  클래스 불변성 체크
	  assert this.count >= 0 : this.count;
	}

```

```
37
38      public void incCount() {
39        changeCount(1);
40        //      changeCount(2); // assertTest#1
41      }
42
43      public void decCount() {
44        changeCount(-1);
45        //      changeCount(-2); // assertTest#2
46      }
47
48      public int getCount() {
49        return this.count;
50      }
51
52
53      public static void main(String[] args) {
54        Counter counter = new Counter(0);
55        counter.incCount();
56        System.out.println("inc counter : " + counter.getCount());
57        counter.decCount();
58        System.out.println("dec counter : " + counter.getCount());
59        counter.decCount();
60        System.out.println("dec counter : " + counter.getCount());
61      }
62
63    }
```

assertTest#1

```
37
38      public void incCount() {
39    //    changeCount(1);
40      changeCount(2); // assertTest#1
41      }
42
```

```
<terminated> Counter [Java Application] C:₩Program Files₩Java₩jre1.8.0_161
Exception in thread "main" java.lang.AssertionError
        at asserttest.Counter.changeCount(Counter.java:20)
        at asserttest.Counter.incCount(Counter.java:40)
        at asserttest.Counter.main(Counter.java:55)
```

```
18
19          // 사전 조건 체크
20          assert val == 1 || val == -1;
21
```

assertTest#2

```
    public void decCount() {
//      changeCount(-1);
        changeCount(-2); // assertTest#2
    }
```

```
<terminated> Counter [Java Application] C:₩Program Files₩Java₩jre1.8.0_161₩b
inc counter : 1
Exception in thread "main" java.lang.AssertionError
        at asserttest.Counter.changeCount(Counter.java:20)
        at asserttest.Counter.decCount(Counter.java:45)
        at asserttest.Counter.main(Counter.java:57)
```

assertTest#3

```
private void changeCount(final int val) {
  // 사후 조건 체크용
  final int preCount = this.count;

  // 사전 조건 체크
  assert val == 1 || val == -1;

  if((this.count + val) < 0) {
    this.count = 0;
  } else {
    this.count += val;
  }

 count = 0; //assertTest#3
  // 사후 조건 체크
  assert (val == 1 && (this.count > preCount)) || (val == -1 && (this.count <= preCount));
```

```
<terminated> Counter [Java Application] C:₩Program Files₩Java₩jre1.8.0_161₩bin₩javaw.exe (2021. 6. 1. 오후 3:06:57)
Exception in thread "main" java.lang.AssertionError
        at asserttest.Counter.changeCount(Counter.java:30)
        at asserttest.Counter.incCount(Counter.java:39)
        at asserttest.Counter.main(Counter.java:55)
```

assertTest#3

```
15⊖     private void changeCount(final int val) {
16        // 사후 조건 체크용
17        final int preCount = this.count;
18
19        // 사전 조건 체크
20        assert val == 1 || val == -1;
21
22        if((this.count + val) < 0) {
23          this.count = 0;
24        } else {
25          this.count += val;
26        }
27
28       // count = 0; //assertTest#3
29        // 사후 조건 체크
30        assert (val == 1 && (this.count > preCount)) || (val == -1 && (this.count <= preCount));
31
32       count = -1 ; //assertTest#4
33        // 클래스 불변성 체크
34        assert this.count >= 0 : this.count;
35      }
36
```

Javadoc Declaration Console

```
<terminated> Counter [Java Application] C:₩Program Files₩Java₩jre1.8.0_161₩bin₩javaw.exe (2021. 6. 1. 오후 3:09:19)
Exception in thread "main" java.lang.AssertionError: -1
        at asserttest.Counter.changeCount(Counter.java:34)
        at asserttest.Counter.incCount(Counter.java:39)
        at asserttest.Counter.main(Counter.java:55)
```

2) strictfp

- strictfp
  - » **strictfp** is a modifier in the Java programming language that restricts floating-point calculations to ensure portability. The strictfp command was introduced into Java with the Java virtual machine (JVM) version 1.2 and is available for use on all currently updated Java VMs.

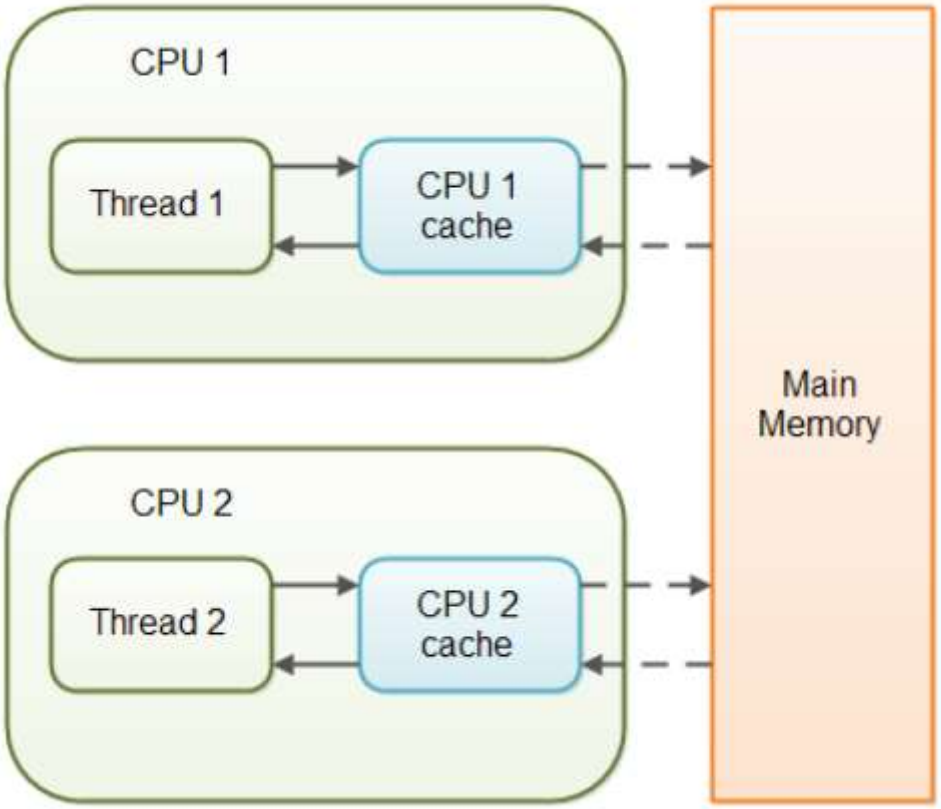
  - » 메소드 또는 클래스 에서 strictfp 수정자를 사용하면 컴파일러는 모든 플랫폼에서 동일한 결과에 대해 Java 스펙을 엄격하게 준수하는 코드를 생성

- strictfp는 클래스, 인터페이스, 비-추상 메서드에 사용될 수 있음.

```
strictfp interface Fooable {}

public strictfp class Foo {}

public class Test {
        public strictfp void echo() {}
}
```

3) volatile

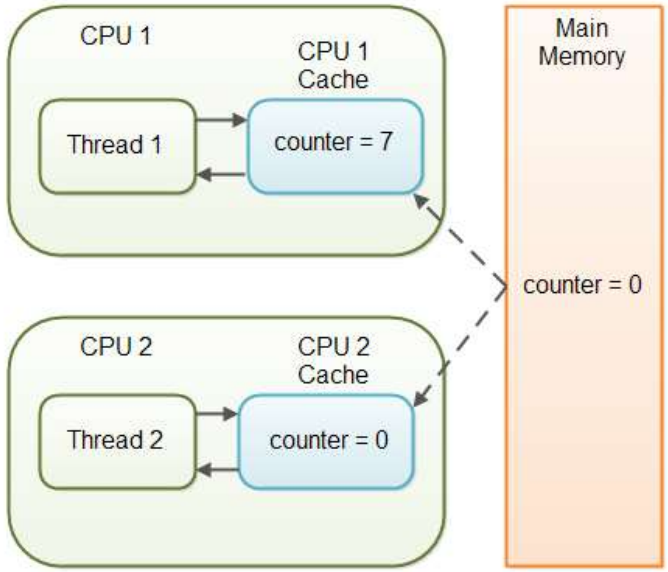
- Java volatile이란?
  - » volatile keyword는 Java 변수를 Main Memory에 저장하겠다라는 것을 명시하는 것
  - » 매번 변수의 값을 Read할 때마다 CPU cache에 저장된 값이 아닌 Main Memory에서 읽음
  - » 또한 변수의 값을 Write할 때마다 Main Memory에 까지 Write함 .



- volatile 변수를 사용하고 있지 않는 MultiThread
  어플리케이션에서는 Task를 수행하는 동안 성능 향상을 위해
  Main Memory에서 읽은 변수 값을 CPU Cache에 저장.
- 만약에 Multi Thread환경에서 Thread가 변수 값을 읽어올 때
  각각의 CPU Cache에 저장된 값이 다르기 때문에 변수 값 불일치
  문제가 발생.

- 예) SharedObject를 공유하는 두 개의 Thread.
  - » Thread-1는 counter 값을 더하고 읽는 연산. (Read & Write)
  - » Thread-2는 counter 값을 읽기만 함 (Only Read)

```
public class SharedObject {
    public int counter = 0;
}
```



- ➔ Thread-1은 counter값을 증가시키고 있지만 CPU Cache에만 반영되어있고 실제로 Main Memory에는 반영이 되지 않음.
  그렇기 때문에 Thread-2는 count값을 계속 읽어오지만 0을 가져오는 문제가 발생.

```
public class SharedObject {
    public volatile int counter = 0;
}
```

- 성능에 미치는 영향
  - » Volatile는 변수의 read와 write를 Main Memory에서 진행.
  - » CPU Cache보다 Main Memory가 비용이 더 크기 때문에 변수 값 일치을 보장해야 하는 경우에만 volatile 사용하는 것이 좋음

4) transient

- Java transient이란?
  - » transient는 Serialize하는 과정에 제외하고 싶은 경우 선언하는 키워드.
  - » 패스워드와 같은 보안정보가 직렬화(Serialize) 과정에서 제외하고 싶은 경우에 적용.
  - » 다양한 이유로 데이터를 전송을 하고 싶지 않을 때 선언.

예) Point Class

```
import java.io.Serializable;

class Point implements Serializable {
  int x, y;
  transient float rho, theta;

  @Override
  public String toString() {
    return String.format("x=%d, y=%d, rho=%f, theta=%f", x, y,
rho, theta);
  }
}
```

format

```
public static String format(String format,
                            Object... args)
```

Returns a formatted string using the specified format string and arguments.

The locale always used is the one returned by `Locale.getDefault()`.

Parameters:

`format` - A format string

`args` - Arguments referenced by the format specifiers in the format string. If there are more arguments than format specifiers, the extra arguments are ignored. The number of arguments is variable and may be zero. The maximum number of arguments is limited by the maximum dimension of a Java array as defined by *The Java™ Virtual Machine Specification*. The behaviour on a null argument depends on the conversion.

Returns:

A formatted string

Throws:

`IllegalFormatException` - If a format string contains an illegal syntax, a format specifier that is incompatible with the given arguments, insufficient arguments given the format string, or other illegal conditions. For specification of all possible formatting errors, see the Details section of the formatter class specification.

Since:

1.5

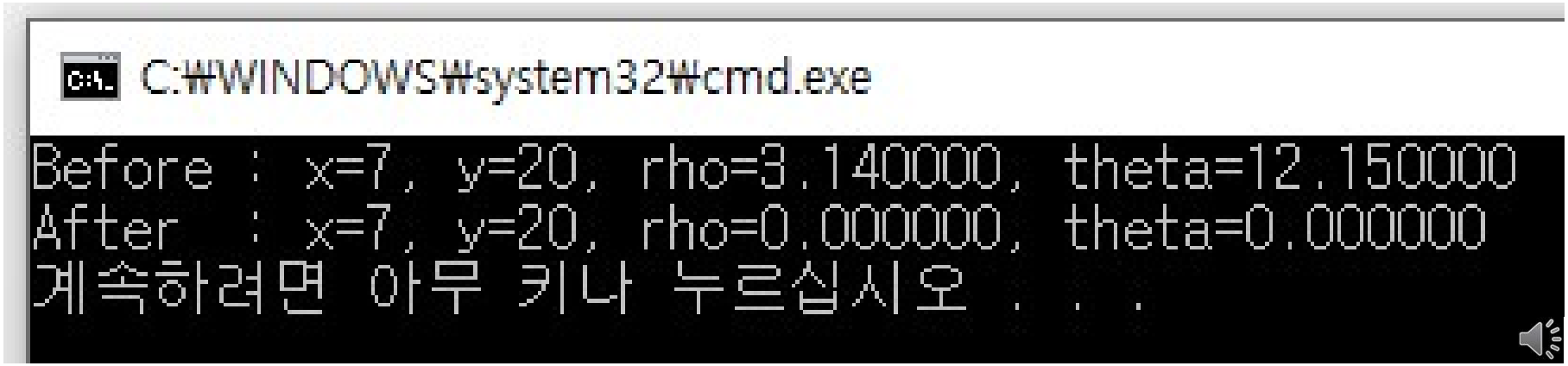
See Also:

`Formatter`

예) TransientSample Class

```
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.ObjectInputStream;
import java.io.ObjectOutputStream;
import java.io.Serializable;

public class TransientSample {
  public static final String FILENAME = "data.ser";

  public static void main(String[] args) throws IOException,
      ClassNotFoundException {

    Point before = new Point();
    before.x = 7;
    before.y = 20;
    before.rho = (float) 3.14;
    before.theta = (float) 12.15;

    writeObject(before);
    Object after = readObject();

    System.out.print("Before : ");
    System.out.println(before);
    System.out.print("After  : ");
    System.out.println(after);
  }
```

```
 private static void writeObject(Object obj) throws IOException {
    FileOutputStream fos = new FileOutputStream(FILENAME);
    ObjectOutputStream oos = new ObjectOutputStream(fos);
    oos.writeObject(obj);
    oos.close();
    fos.close();
  }


  private static Object readObject() throws IOException, ClassNotFoundException
{
    FileInputStream fis = new FileInputStream(FILENAME);
    ObjectInputStream ois = new ObjectInputStream(fis);
    Object result = ois.readObject();
    ois.close();
    fis.close();
    return result;
  }

}
```



예) Point Class

```
import java.io.Serializable;

class Point implements Serializable {
  int x, y;
 float rho, theta;  // Transient 를 쓰지 않을때

  @Override
  public String toString() {
    return String.format("x=%d, y=%d, rho=%f, theta=%f", x, y,
rho, theta);
  }
}
```