JDK 7 출시 기념 (2011.7)
JDK 7 소개 #2 Project Coin

김용환

knight76.tistory.com

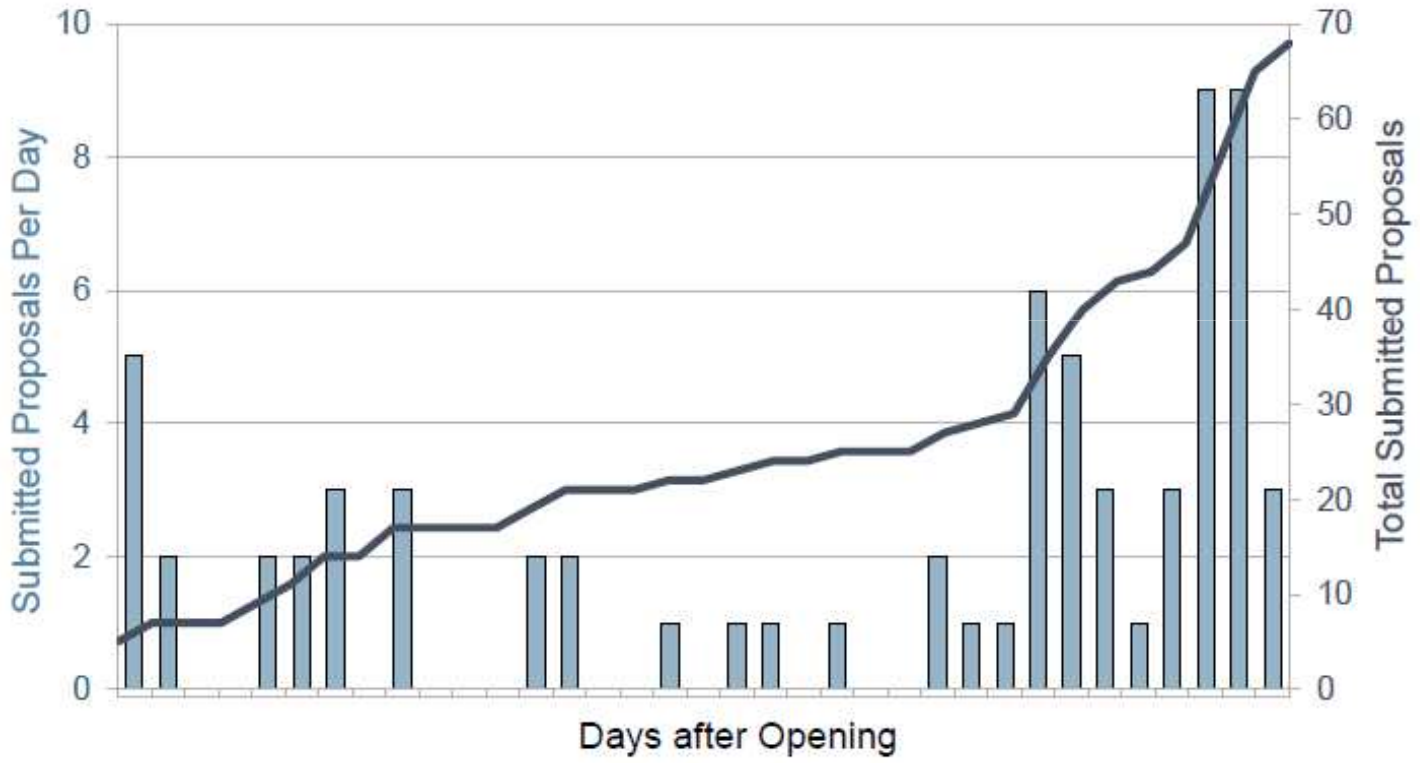
Knight76 at gmail.com

Project Coin

역사

- Project Leader인 Joseph D. Darcy(오라클 직원)이 IDEA를 낸 것으로 알려짐
- 2 27, 2009 부터 3 30, 2009일까지 개발자로부터 요구사항을 받음(open call)
  - 70개 정도 받아서 일부만 jdk 7에 반영, 일부는 이후로 미룸
- Language에 영향을 주기 때문에, 'The Java programming-language compiler (javac) group' 에서 관리
- 2011 OSCON에서 공식 발표 예정

Proposals per day



출처 : http://www.oracle.com/us/technologies/java/project-coin-428201.pdf

정리된 Proposal

- http://wikis.sun.com/display/ProjectCoin/2009ProposalsTOC?focusedCommentId=183894025#comment-183894025

Proposals Accepted for Inclusion in JDK 7

On August 28, 2009, Joe Darcy announced the final selection of proposals accepted for JDK 7 in this blog entry.

Two proposals, "Binary Literals" and "Underscores in Numbers" will be combined, possibly with additional support for unsigned literals.

Another two proposals, "Collection Literals" and "Indexing access syntax for Lists and Maps" will also be merged and included, "...assuming the technical problems are resolved." (Darcy)

Proposal	By	On	Thread
Strings in Switch (search)	Joseph D. Darcy	February 27, 2009	Mail Thread
Automatic Resource Management (search)	Josh Bloch	February 27, 2009	Mail Thread
Improved Type Inference for Generic Instance Creation (search)	Jeremy Manson	February 28, 2009	Mail Thread
Simplified Varargs Method Invocation (search)	Bob Lee	March 6, 2009	Mail Thread (V1) (V2)
Collection Literals (search)	Joshua Bloch	March 30, 2009	Mail Thread
Indexing access syntax for Lists and Maps (search)	Shams Mahmood	March 30, 2009	Mail Thread
Language support for JSR 292 (search)	John Rose	March 28, 2009	Mail Thread
Binary Literals (search)	Derek Foster	March 25, 2009	Mail Thread (V1) (V2)
Underscores in numbers (search)	Derek Foster	March 31, 2009	Mail Thread

Additional Proposals

Proposal	By	On	Thread
Block Expressions for Java (search)	Neal Gafter	February 28, 2009	Mail Thread
Improved Exception Handling for Java (search)	Neal Gafter	February 28, 2009	Mail Thread

Reference

- 오라클 블로그
  - http://blogs.oracle.com/main/tags/projectcoin
  - 프로젝트 리더 Darcy의 블로그 http://blogs.oracle.com/darcy
- 메일링 리스트
  - http://mail.openjdk.java.net/mailman/listinfo/coin-dev
- 위키
  - http://wikis.sun.com/display/ProjectCoin/Home
- Draft (강추)
  - JSR 334: “Small Enhancements to the Java™Programming Language” http://www.jcp.org/en/jsr/detail?id=334

JDK7 채택

- Strings in switch
- Binary Integer
- Underscore Characters in Numeric Literals
- Improved Type Inference for Generic Instance Creation (diamond)
- Using Non-Reifiable Parameters with Varargs Methods

  (Simplified Varargs Method Invocation)
- Multi-catch and more precise rethrow
- Automatic Resource Management (try-with-resources)

다음 기회에

- Multiple line : “””
- For-each
- List, map을 배열처럼
- String 을 if Null 하는 부분 간단하게
- 등등

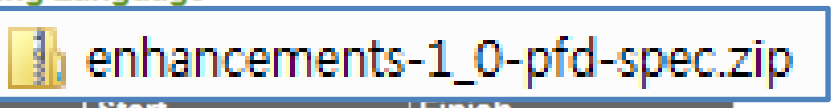
JSR 334-Project Coin

JSR | Update | Expert Group

Summary | Proposal | Detail (Summary & Proposal)

JSRs: Java Specification Requests

JSR 334: Small Enhancements to the Java™ Programming Language



Stage	Access	Start	Finish
Final Approval Ballot		05 Jul, 2011	18 Jul, 2011
Proposed Final Draft	Download page	24 Jun, 2011	
Public Review Ballot	View results	19 Apr, 2011	25 Apr, 2011
Public Review	Download page	23 Mar, 2011	25 Apr, 2011
Early Draft Review	Download page	11 Jan, 2011	10 Feb, 2011
Expert Group Formation		07 Dec, 2010	
JSR Review Ballot	View results	16 Nov, 2010	06 Dec, 2010

Strings in switch

소스

```
public class Test {
        public static void main(String[] args) {
                String str = "surname";
                switch (str) {
                case "name":
                        System.out.println("name");
                        break;
                case "surname":
                        System.out.println("surname");
                        break;
                case "forename":
                        System.out.println("forname");
                        break;
                }
        }
}
```

결과

```
surname
```

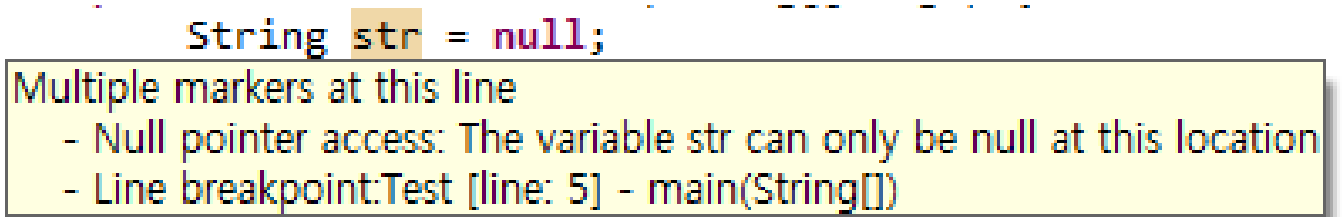
Strings in switch

- String 의 값이 null이면?

```
String str = null;
switch (str) {
```

```
Exception in thread "main" java.lang.NullPointerException
        at Test.main(Test.java:5)
```

  - Eclipse에서의 자동 빌드에서는 다음과 같이 Warning 알려줌



Binary Integer

```
int binVal = 0b11010; // binary, int 26
System.out.println(binVal);

byte nybbles = 0b00100101; // binary, int 37
System.out.println(nybbles);
```

```
26
37
```

Underscore Characters in Numeric Literals

- 숫자 사용시 '_'를 사용하여 구분지을 수 있게 함. 우리가 일상시에 사용하는 ',' 단위로 쓰는 것처럼 혼동되지 않게 하기 위함
  - 기존 코드 long number = 1000000; // million
  - '_' 를 사용하여 가독성을 높임

```
long creditCardNumber = 1234_5678_9012_3456L;
long socialSecurityNumber = 999_99_9999L;
float pi = 3.14_15F;
long hexBytes = 0xFF_EC_DE_5E;
long hexWords = 0xCAFE_BABE;
long maxLong = 0x7fff_ffff_ffff_ffffL;
long bytes = 0b11010010_01101001_10010100_10010010;
```

Underscore Characters in Numeric Literals

- 주의 사항
  - '_' 는 맨 처음과 맨 마지막에 올 수 없다.
    - 0b01010_ (X)
    - _0b1111 (X)
  - Decimal point(.) 주위에서는 쓸 수 없다.
    - float pi2 = 3._1415F; (X)
    - float pi2 = 3_.1415F; (X)
  - 0x와 같이 진수를 표시하는 예약어 주위에 '_'를 사용할 수 없다.
    - byte x2 = _0b11; (X)
    - int x1 = 0_x52; (X)

Underscore Characters in Numeric Literals

- 주의 사항
  - F나 L 같이 floating, long 타입의 suffix 뒤에 사용할 수 없다.
    - float x2 = 0x11F_; (X)
  - 숫자 사이에서만 사용 가능
    - Int x = 0x_52 ; (X)
    - 예외 : float x2 = 0x11_F; (0)
- 특별하게 사용 가능
    - float x2 = 0x11_____F;
    - int x3 = 5_____2;

Improved Type Inference for Generic Instance Creation (diamond)

- 특정 Instance의 Type을 Compiler가 알아서 찾아주는 것을 Inference(추론)라고 함
- 즉 Type Inference for Generic Instance Creation은 Compiler가 알아서 타입을 추론해 줌. 중복 코딩 방지 기능.

```
Map<String, List<String>> myMap = new HashMap<String, List<String>>();
```

'<>' 를 diamond 연산자라 한다.

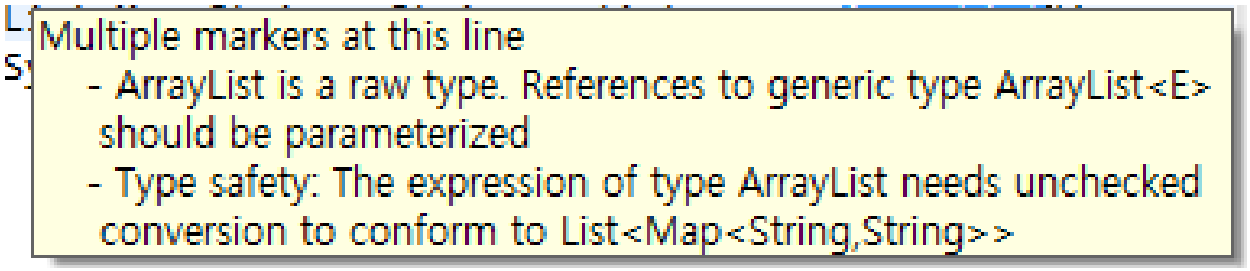
```
Map<String, List<String>> myMap = new HashMap<>();
```

* Warning이 발생되지 않음

Improved Type Inference for Generic Instance Creation (diamond)

- HashMap<> 사용을 안할 때?
  - Unchecked conversion warning!

```
Map<String, List<String>> myMap = new HashMap();
```



Improved Type Inference for Generic Instance Creation (diamond)

- 주의 사항
  - Type inference를 모를 상황에서 그냥 넣으면 에러 발생

```
List<String> list = new ArrayList<>();
list.add("A");
list.addAll(new ArrayList<>()); // compile error
```

  - 다음과 같이 사용해야 문제 발생 안함

```
List<String> list = new ArrayList<>();
list.add("A");
List<? extends String> list2 = new ArrayList<>(); // List<String> 선언해도 됨
list.addAll(list2); // no compile error
```

Improved Type Inference for Generic Instance Creation (diamond)

- 다른 언어를 보았을 때, 상황에 맞게 써야지. 무작정 쓰면 Type Inference 가 기대에 맞게 오지 않을 수 있음
- Java 언어를 만들 때부터 Compiler에 Type Inference 를 넣지 않았음을 생각해볼 필요는 있음

Improved Type Inference for Generic Instance Creation (diamond)

- 생성시 주의 사항

```
class MyClass<X> {
  <T> MyClass(T t) { }
}
..
public static void main(String[] args) {
    new MyClass<Integer>(""); // no compile error
    MyClass<Integer> myObject1 = new MyClass<>("");  // no compile error
    MyClass<Integer> myObject2 = new <String> MyClass<>("");  // compiler error
}
```

Explicit type arguments cannot be used with '<>' in an allocation expression

  - 첫번째 new MyClass<Integer>("")는 T가 String, X는 Integer로 Type Inference가 되어서 문제가 없음
  - 두번째도 역시 첫번째와 같이 inference 됨
  - 세번째는 컴파일 에러가 발생, T는 String, X는 Integer로 inference 되었는데, <>를 쓰면서(추론) 동시에 명시적으로 String 타입을 T로 인식하게 하는 것은 컴파일 에러가 남

Improved Type Inference for Generic Instance Creation (diamond)

- 어디서 Diamond를 쓸지에 대한 블로그글
  - http://stuartmarks.wordpress.com/2011/01/24/where-can-diamond-be-used/
  - field or local variable initializer (61%)
    - private static Map<ClassLoader, Map<List<String>, Object>> loaderToCache = new WeakHashMap<>();
  - right-hand side of assignment statement (33%)
    - List<Permission> perms;
perms = new ArrayList<>();
  - method argument (4%)
    - List<List<Attribute.Layout>> attrDefs = ...;
attrDefs.set(i, new ArrayList<>(...));
  - return statement (2%)
    - public Enumeration<URL> getResources(String name) {
return new CompoundEnumeration<>(tmp);
}

Using Non-Reifiable Parameters with Varargs Methods

- Type Erasure 개념
  - 객체를 초기화시, compiler가 클래스 또는 메소드의 type 파타미터의 generic 정보를 없애는 것을 말함
  - 런타임시 type 정보를 잃어 버린다. (decompile 해보면 알게됨) 그래서 type 오류를 compile 시점에서 처리함
  - Generic를 쓰지 않는 Legacy code와의 호환성을 위한 것 (binary compatibility)

Using Non-Reifiable Parameters with Varargs Methods

- Reifiable type
  - 영한 사전에 없는 단어 (헐!) http://java.sun.com/docs/books/jls/third_edition/html/typesValues.html#112581
  - 런타임에도 완벽하게 타입정보가 계속 남아있는 타입 (Types that are completely available at run time)
  - String, Number
- Non-reifiable type
  - Type erasure에 의해서 타입 정보가 손실되는 타입
  - A non-reifiable type is a type that is *not* completely available at runtime
  - ArrayList<Number>, List<String>
- Non-Reifiable Parameters with Varargs Methods
  - 타입 정보가 손실되는 Varargs 타입을 가진 메소드

Using Non-Reifiable Parameters with Varargs Methods

- 그 동안의 문제
  - Unchecked warning이 컴파일시 발생했지만, 이것이 결국은 ClassCastException을 발생

```
public static void main(String[] args) {
    List l = new ArrayList<Number>();
    List<String> ls = l;       // unchecked warning
    l.add(0, new Integer(42)); // another unchecked warning
    String s = ls.get(0);      // At runtime, ClassCastException 발생
}
```

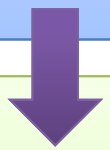
```
Exception in thread "main" java.lang.ClassCastException: java.lang.Integer
cannot be cast to java.lang.String
at Test.main(Test.java:11)
```

Using Non-Reifiable Parameters with Varargs Methods

- Decompile 된 코드 (javac로 컴파일된 class)

원래

```
public static void main(String[] args) {
    List l = new ArrayList<Number>();
    List<String> ls = l;       // unchecked warning
    l.add(0, new Integer(42)); // another unchecked warning
    String s = ls.get(0);      // At runtime, ClassCastException 발생
}
```



Type Erasure에 의해서
Non-reifiable Type이 모두
Reifiable type으로 변경

디컴파일

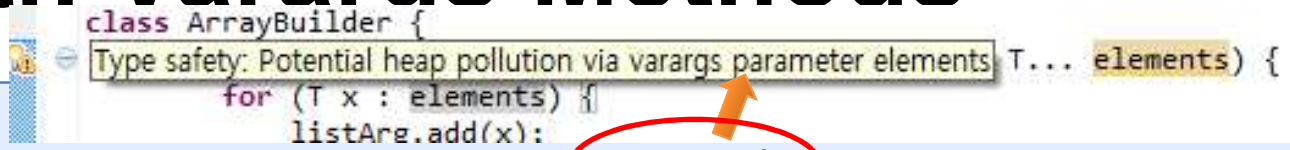
```
public static void main(String[] args) {
    List l = new ArrayList();
    List ls = l;
    l.add(0, new Integer(42));
    String s = (String)ls.get(0);
}
```

Integer의 값을 String을 classcasting하면서 Exception 발생

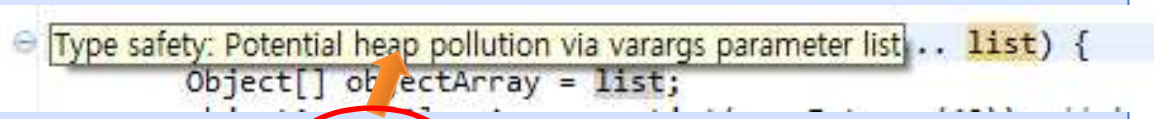
Using Non-Reifiable Parameters with Varargs Methods

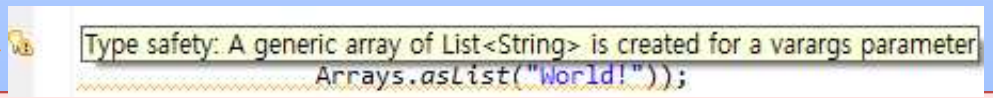
- Non-Reifiable Parameter with varargs method 란?
  - void faultyMethod(List<String>... List) Warning - Type safety: Potential heap pollution via varargs parameter list
  - Type Safety 컴파일 warning이 발생하고, 그냥 볼 수 밖에 없는 상태
  - List<String> 파라미터를 전달해도 Runtime때는 List일 뿐

Using Non-Reifiable Parameters with Varargs Methods




```
class ArrayBuilder {
        public static <T> void addToList(List<T> listArg, T... elements) {
                for (T x : elements) {
                        listArg.add(x);
                }
        }

        public static void faultyMethod(List<String>... list) {
                Object[] objectArray = list;
                objectArray[0] = Arrays.asList(new Integer(42));   // ClassCastException
                String str = list[0].get(0);
        }
}

public class Test {
        public static void main(String[] args) {
                List<String> stringListA = new ArrayList<String>();
                ArrayBuilder.addToList(stringListA, "Seven", "Eight", "Nine");
                ArrayBuilder.faultyMethod(Arrays.asList("Hello!"), Arrays.asList("World!"));
        }
}
```





Exception in thread "main" java.lang.ClassCastException: java.lang.Integer cannot be cast to java.lang.String

Using Non-Reifiable Parameters with Varargs Methods

- JDK 5,6 에서는 컴파일러는 ArrayBuilder.addToList()가 호출한 곳에서만 Warning 발생. 선언부에서는 Warning 발생하지 않음

- JDK7부터는 호출한 곳, 선언한 곳 모두 Warning을 발생. 선언한 메소드에 @SafeVarargs Annotation을 사용하면 호출한 곳에서는 Warning 발생 안함

Using Non-Reifiable Parameters with Varargs Methods

- 오해의 소지
  - ClassCastException은 잘 못 쓰면 어쩔 수 없이 생길 수 밖에 없음.
    (Potential Vulnerabilities)
  - JDK7에서는 선언부 관점에서도 Warning을 표시해서 좀 더 보기 편하게 하자는 의미

Using Non-Reifiable Parameters with Varargs Methods

```
class ArrayBuilder {
        @SafeVarargs  // Warning 발생 안함
            public static <T> void addToList(List<T> listArg, T... elements) {
                        for (T x : elements) {
                                    listArg.add(x);
                        }
            }

        @SafeVarargs  // Warning 발생 안함
            public static void faultyMethod(List<String>... list) {
                        Object[] objectArray = list;
                        objectArray[0] = Arrays.asList(new Integer(42));   // ClassCastException
                        String str = list[0].get(0);
            }
}

public class Test {
            public static void main(String[] args) {
                        List<String> stringListA = new ArrayList<String>();
                        ArrayBuilder.addToList(stringListA, "Seven", "Eight", "Nine");
                        ArrayBuilder.faultyMethod(Arrays.asList("Hello!"), Arrays.asList("World!"));
                    // Warning 발생 안함
            }
}
```

Using Non-Reifiable Parameters with Varargs Methods

- JDK7 compiler는 문제의 원인이 될 만한 곳에 Warning을 출력함!!

```
class ArrayBuilder {
        @SafeVarargs
            public static <T> void addToList(List<T> listArg, T... elements) {
                        for (T x : elements) {
                                    listArg.add(x);
                        }
            }

        @SafeVarargs
            public static void faultyMethod(List<String>... list) {
                        Object[] objectArray = list;
                        objectArray[0] = Arrays.asList(new Integer(42));   // ClassCastException
                        String str = list[0].get(0);
            }
}
```

Test.java:15: warning: [varargs] Varargs method could cause heap pollution from non-reifiable varargs parameter list

```
        Object[] objectArray = list;
```

Using Non-Reifiable Parameters with Varargs Methods

- 일반 메소드에서 다 사용
  - @SuppressWarnings({"unchecked", "varargs"})
- Static (or final) & varargs & 재정의 아닌 메소드
  - @SafeVarargs

```
class ArrayBuilder {
    @SafeVarargs
    public static <T> void addToList(List<T> listArg, T... elements) {
        for (T x : elements) {
            listArg.add(x);
        }
    }

    @SuppressWarnings({"unchecked", "varargs"})
    public <T> void vfg(List<T> listArg, T... elements) {
        for (T x : elements) {
            listArg.add(x);
        }
    }

    @SuppressWarnings({"unchecked", "varargs"})
    public static <T> void bb(List<T> listArg, T... elements) {
        for (T x : elements) {
            listArg.add(x);
        }
    }
}
```

* STS 2.7& Eclipse jdk7 compiler plugin(beta)에서는 최신 것을 반영하지 않아 warning처럼 보이나, Oracle Jdk7 compiler를 이용하면 warning 발생하지 않음

Using Non-Reifiable Parameters with Varargs Methods

- jdk 7 compiler 이용

```
C:\test\src>"c:\Program Files\Java\jdk1.7.0\bin\javac.exe" -Xlint Test.java
Test.java:7: warning: [unchecked] Possible heap pollution from parameterized vararg type T
        public static <T> void addToList(List<T> listArg, T... elements) {
                                         ^
  where T is a type-variable:
    T extends Object declared in method <T>addToList(List<T>,T...)
1 warning


C:\test\src>"c:\Program Files\Java\jdk1.7.0\bin\javac.exe" -Xlint:varargs
 Test.java
Note: Test.java uses unchecked or unsafe operations.
Note: Recompile with -Xlint:unchecked for details.
```

Using Non-Reifiable Parameters with Varargs Methods

- JDK7 소스에 반영 (4군데)

```
// JDK7의 java.util.Arrays의 asList(T… A) 메소드

 @SafeVarargs
   public static <T> List<T> asList(T... a) {
       return new ArrayList<>(a);
   }
```

<T> List<T> Arrays.asList(T... a)

<T> booleanCollections.addAll(Collection<? super T> c, T... elements)

<E extends Enum<E>> EnumSet<E> EnumSet.of(E first,E... rest)

void javax.swing.SwingWorker.publish(V... chunks)

Multi-catch and more precise rethrow

- Multi-catch
  - 하나의 Exception Handler로 여러 개의 Exception을 하나로 처리함
  - Catch 문 안에서 '|' 를 이용해서 여러 개의 Exception 을 나열가능, rethrow도 가능
  - 부모-자식 관계에서는 사용 불가능

```
class Throw {
        public static void throwIOE() throws IOException {
                throw new IOException("test");
        }

        public static void throwSQE() throws SQLException {
                throw new SQLException("test");
        }
}

public class Test {
        public static void main(String[] args) {
                try {
                        Throw.throwIOE();
                        Throw.throwSQE();
                } catch (IOException | SQLException e) {
                        System.err.println(e);
                    throw e;  // 가능
                }
        }
}
```

Multi-catch and more precise rethrow

JDK7

```
try {
        Throw.throwIOE();
        Throw.throwSQE();
} catch (IOException | SQLException e) {
        System.err.println(e);
}
```

디컴파일

```
try
{
        Throw.throwIOE();
        Throw.throwSQE();
}
catch(Exception e)
{
        System.err.println(e);
}
```

Multi-catch and more precise rethrow

- Multi-catch안에서 부모-자식 관계의 Exception 사용시 Errror!

```
class ParentException extends Exception {}
class ChildException extends ParentException {}

class Throw {
        public static void throwIOE() throws ParentException {
                throw new ParentException();
        }
        public static void throwSQE() throws ChildException {
                throw new ChildException();
        }
}

public class Test {
        public static void main(String[] args) {
                try {
                        Throw.throwIOE();
                        Throw.throwSQE();
                } catch (ParentException | ChildException e) {
                        System.err.println(e);
                }
        }
}
```

- 해결방법
  - Catch(Exception e)
  - Parent Exception만 잡기

```
error: Alternatives in a multi-catch statement cannot be
related by subclassing
                } catch (ParentException | ChildException e) {
                                        ^
  Alternative ChildException is a subclass of alternative
ParentException
```

Multi-catch and more precise rethrow

- catch 절에서 exception을 throw할 때, 메소드에서 정의하는 타입을 결정하고, 실제 Exception의 타입을 전달이 가능하다.
- 컴파일러가 관련 코드를 변경하지 않음

```
class ExceptionB extends Exception{}

public static void testReThrow() throws ExceptionB {
        …
        } catch (final Exception ex) {  // Throwable 도 사용 가능
                throw ex;
        }
}
```

Multi-catch and more precise rethrow

- 기존의 컴파일방식이 달라 하위 호환 (binary compatibility)이 되지 않음

```
c:\test>"c:\Program Files\Java\jdk1.7.0\bin\javac.exe" -source 1.6 Test.java

warning: [options] bootstrap class path not set in conjunction with -source 1.6
Test.java:23: error: unreported exception Throwable; must be caught or declared
to be thrown
                    throw ex;
                    ^
1 error
1 warning
```

Multi-catch and more precise rethrow

- JDK7에서만 아래와 같은 코드가 컴파일 이 됨
- 주의사항
  - 하위 호환성이 중요하다면, 쓰지 말아야 하는 코드

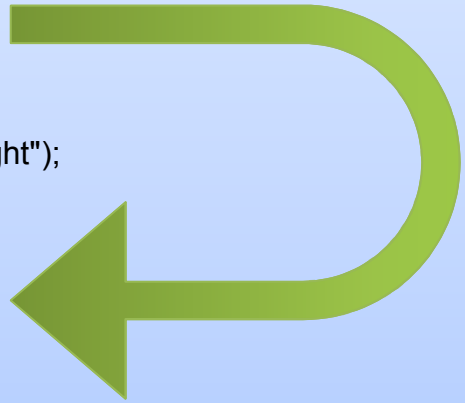
```
class ExceptionA extends Exception{}
class ExceptionB extends Exception{}

class A {
        public static void throwExceptionA() throws ExceptionA {}
        public static void throwExceptionB() throws ExceptionB {}
        public static void testReThrow() throws ExceptionB {
                try {
                        throwExceptionA();
                        throwExceptionB();
                } catch (final ExceptionA exa) {
                        //...
                } catch (final Exception ex) {
                        throw ex;
                }
        }
}
```

Multi-catch and more precise rethrow

```
class ExceptionA extends Exception {};
class ExceptionB extends Exception {};
class A {
        public static void testReThrow() throws ExceptionB {
                final int a = 0;
                try {
                        if (a == 0) {
                                throw new ExceptionA();
                        }
                        if (a == 1) {
                                throw new ExceptionB();
                        }
                } catch (final ExceptionA exa) {
                        System.out.println("ExceptionA was caught");
                } catch (final Exception ex) {
                        System.out.println(ex.getClass()
                        + " was thrown, caught and rethrown");
                        throw ex;
                }
        }
}
…
try {
   A.testReThrow();
} catch (ExceptionB e) {
   e.printStackTrace();
}
```

JDK6에서는 compile failed
- throws Exception으로 변경
- throw하는 Exception 타입!

JDK7에서는 compile OK
- 실제 Exception이 발생하는 Instance의 타입이 ExceptionB



<결과>
ExceptionA was caught

try-with-resources

- Resource ?
  - Java 프로그램이 종료하고 나서 close해야 할 것들(object)
  - 보통 resource라 함은 이미지, 리스너, 스트림, 소켓, 파일, 채널 등등이 있지만, 여기서는 DB관련클래스, 스트림, 소켓, 채널을 의미함

java.lang

Interface AutoCloseable

All Known Subinterfaces:

AsynchronousByteChannel, AsynchronousChannel, ByteChannel, CachedRowSet, CallableStatement, Channel, Clip, Closeable, Connection, DataLine, DirectoryStream<T>, FilteredRowSet, GatheringByteChannel, ImageInputStream, ImageOutputStream, InterruptibleChannel, JavaFileManager, JdbcRowSet, JMXConnector, JoinRowSet, Line, MidiDevice, MidiDeviceReceiver, MidiDeviceTransmitter, Mixer, MulticastChannel, NetworkChannel, ObjectInput, ObjectOutput, Port, PreparedStatement, ReadableByteChannel, Receiver, ResultSet, RMIConnection, RowSet, ScatteringByteChannel, SecureDirectoryStream<T>, SeekableByteChannel, Sequencer, SourceDataLine, StandardJavaFileManager, Statement, SyncResolver, Synthesizer, TargetDataLine, Transmitter, WatchService, WebRowSet, WritableByteChannel

All Known Implementing Classes:

AbstractInterruptibleChannel, AbstractSelectableChannel, AbstractSelector, AsynchronousFileChannel, AsynchronousServerSocketChannel, AsynchronousSocketChannel, AudioInputStream, BufferedInputStream, BufferedOutputStream, BufferedReader, BufferedWriter, ByteArrayInputStream, ByteArrayOutputStream, CharArrayReader, CharArrayWriter, CheckedInputStream, CheckedOutputStream, CipherInputStream, CipherOutputStream, DatagramChannel, DatagramSocket, DataInputStream, DataOutputStream, DeflaterInputStream, DeflaterOutputStream, DigestInputStream, DigestOutputStream, FileCacheImageInputStream, FileCacheImageOutputStream, FileChannel, FileImageInputStream, FileImageOutputStream, FileInputStream, FileLock, FileOutputStream, FileReader, FileSystem, FileWriter, FilterInputStream, FilterOutputStream, FilterReader, FilterWriter, Formatter, ForwardingJavaFileManager, GZIPInputStream, GZIPOutputStream, ImageInputStreamImpl, ImageOutputStreamImpl, InflaterInputStream, InflaterOutputStream, InputStream, InputStream, InputStream, InputStreamReader, JarFile, JarInputStream, JarOutputStream, LineNumberInputStream, LineNumberReader, LogStream, MemoryCacheImageInputStream, MemoryCacheImageOutputStream, MLet, MulticastSocket, ObjectInputStream, ObjectOutputStream, OutputStream, OutputStream, OutputStream, OutputStreamWriter, Pipe.SinkChannel, Pipe.SourceChannel, PipedInputStream, PipedOutputStream, PipedReader, PipedWriter, PrintStream, PrintWriter, PrivateMLet, ProgressMonitorInputStream, PushbackInputStream, PushbackReader, RandomAccessFile, Reader, RMIConnectionImpl, RMIConnectionImpl_Stub, RMIConnector, RMIIIOPServerImpl, RMIJRMPServerImpl, RMIServerImpl, Scanner, SelectableChannel, Selector, SequenceInputStream, ServerSocket, ServerSocketChannel, Socket, SocketChannel, SSLServerSocket, SSLSocket, StringBufferInputStream, StringReader, StringWriter, URLClassLoader, Writer, XMLDecoder, XMLEncoder, ZipFile, ZipInputStream, ZipOutputStream

- Try with resources ?
  - Resource를 편하게 close 하게 해서 여유로운 어플 상황을 누르게 함

try-with-resources

JDK6

```
static String readFirstLineFromFileWithFinallyBlock(String path) throws
IOException {
  BufferedReader br = new BufferedReader(new FileReader(path));
  try {
    return br.readLine();
  } finally {
    if (br != null) br.close();
  }
}
```

try 키워드만 쓰면 프로그램 종류후, BufferedReader에 대한 close 작업이 이루어 짐

JDK7

```
static String readFirstLineFromFile(String path) throws IOException {
  try (BufferedReader br = new BufferedReader(new FileReader(path))) {
    return br.readLine();
  }
}
```

try-with-resources

- Try with resource 문법 (java7 language spec 14.20.3)

```
TryWithResourcesStatement:
    try ResourceSpecification
       Block
    Catches opt
    Finally opt

ResourceSpecification:
( Resources ;opt )

Resources:
Resource
Resource ; Resources

Resource:
  VariableModifiersopt Type VariableDeclaratorId = Expression
```

try-with-resources

- 사실은 이것은 JDK7 컴파일러의 비밀!!!

원래

```
static String readFirstLineFromFile(String path) throws IOException {
  try (BufferedReader br = new BufferedReader(new FileReader(path))) {
    return br.readLine();
  }
}
```

디컴파일해보면, JDK6에서 해야 하는 코드는 똑 같다.

디컴파일

```
Throwable throwable = null;
Object obj = null;
try {
    BufferedReader br = new BufferedReader(new
FileReader("aa"));
    try {
      br.readLine();
    }
    catch(Throwable throwable) {
      if(br != null)  br.close();
          throw throwable;
      }
   if(br != null)  br.close();
}
```

```
catch(Throwable throwable1) {
    if(throwable == null)
        throwable = throwable1;
    else
        if(throwable != throwable1)
         throwable.addSuppressed(throwable1);
         throw throwable;
}
```

try-with-resources

- Try는 아무 클래스에서 쓸 수 있나?
  - 아니. Resource가 java.lang.AutoCloseable 인터페이스를 상속받아야 한다.
- java.lang.AutoCloseable ?
  - 더 이상 쓰지 않으면 close되어야 하는 자원

```
public interface AutoCloseable {
    void close() throws Exception;
}
```

try-with-resources

- Suppressed Exception
  - 파일(소켓) 관련 처리시에 아래 두 장소에서 exception 날 가능성이 존재
    - 로직, BufferReader.readline()
    - 자원 종료, Close()
  - 하나면 보여주기엔 아까움 (쌍쌍바?)


```
<Exception 발생>
MyException: Exception in work()
at AutoClose.work(Test.java:57)
at Test.main(Test.java:40)
Suppressed: java.lang.RuntimeException: Exception in close()
        at AutoClose.close(Test.java:52)
        at Test.main(Test.java:41)
```

try-with-resources

```
public class AutoClose implements AutoCloseable {
        @Override
        public void close() {
                System.out.println(">>> close()");
                throw new RuntimeException("Exception in close()");
        }
        public void work() throws MyException {
                System.out.println(">>> work()");
                throw new MyException("Exception in work()");
        }
        public static void main(String[] args) throws Exception {
                try (AutoClose autoClose = new AutoClose()) {
                        autoClose.work();
                } catch (MyException e) {
                        e.printStackTrace();
                }
        }
}

class MyException extends Exception {
        public MyException() {
                super();
        }
        public MyException(String message) {
                super(message);
        }
}
```

```
<Exception 발생>
MyException: Exception in work()
at AutoClose.work(Test.java:57)
at Test.main(Test.java:40)
Suppressed: java.lang.RuntimeException: Exception in close()
        at AutoClose.close(Test.java:52)
        at Test.main(Test.java:41)
```

try-with-resources

- Suppressed Exception
  - java.lang.Throwable 클래스에 영향을 미침

To be continued #3