upstate.edu

Lesson 1: Java Reflection I Formats ISUNY Upstate Medical University

27-34 minutes

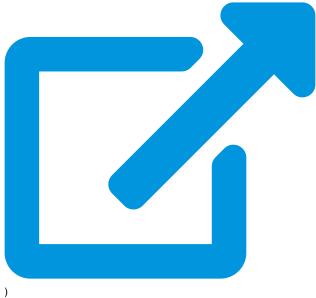
What is Java Reflection?

• Java Reflection Tutorial



 Reflection allows programmatic access to information about the fields, methods and constructors of loaded classes, and the use of reflected fields, methods, and constructors to operate on their underlying counterparts, within security restrictions.

(See java.lang.reflect



• In layman's terms, reflection enables us to conjure up Java objects out of the thin air and/or call methods defined in some Java class in the context of Velocity

Why Java Reflection?

- There are so many useful objects that can be created in the Velocity context, each coming with a set of methods; we want to be able to access these objects
- Example: we want to be able to use Java trees to drive presentation (see <u>Velocity Advanced Course</u>), hence we want to be able to create Java trees
- There are also many useful static methods defined in Java, and we want to be able to call them
- In Velocity, to call either instance methods or static methods without reflection, we must have an object of the right class first
- Java objects have to be instantiated by invoking constructors
- Velocity does not support the new keyword, and hence no constructors of any class can be called directly
- To invoke a static method of a class, we still need an object instantiated from that class (see how do i convert from String to int (or any other number type)?



-); but see below for a way to invoke static methods without creating objects (a way that may cause unexpected problems)
- Without the capability of instantiating Java objects, there are many things that we cannot do, or at least not as easily
- There are also classes like java.lang.Math that do not define
 any constructors at all, and there is no way to call static methods
 defined in this class without using reflection (again, see an
 exception below)

Some Tasks that We Cannot Do (Easily) without Reflection

- · Creating an object to represent tomorrow
- · Creating a calendar of the current month
- Parsing a String into an int, and preventing a possible NumberFormatException
- Even when working with reflection, we may still need to use reflection; for example, to retrieve a method (by calling

- getMethod) by passing in an array of parameters
- Create a Java tree to store the structure of a folder in Cascade or the XHTML markup structure of a page
- Generate basic API documentation of any Java classes, including Cascade API classes

Introducing the java.lang.Class Class

- The key to Java reflection is the java.lang.Class class
- This class provides a static method forName, which accepts a class name as a String, and returns a java.lang.Class object representing the named class
- Out of this java.lang.Class object, methods like getConstructors, getMethods, and getFields can be called
- However, to be able to call forName, we need to start with a java.lang.Class Object
- Fortunately, there are many objects in the Velocity context, and out of any of these objects, the getClass method or Class property returns a java.lang.Class object
- We have been using \$x.Class.Name to reveal class name; it is this \$x.Class object that we need
- We can use, for example, \$.Class
- To show the class name of this \$_.Class object, try
 \$_.Class.Class.Name

A Critical Distinction

- It is critical to the understanding of reflection by making a distinction between a java.lang.Class object and the class it represents
- The type of a java.lang.Class object is of course java.lang.Class
- However, such an object represents a class, possibly (and most likely) a class other than java.lang.Class
- For example, a java.lang.Class object representing
 java.lang.Math contains at least two pieces of class name
 information: the type of the object itself (i.e., java.lang.Class)
 and the class it represents (in this case, java.lang.Math)
- . Do not be confused by these two classes involved at the same time
- We will see this possible confusion when we consider getConstructor and getMethod

Important Methods of java.lang.Class

- static Class<?> forName(String className)
- Constructor<T> getConstructor(Class<?>... parameterTypes)
- Constructor<?>[] getConstructors()
- Field getField(String name)
- Field[] getFields()
- Method getMethod(String name, Class<?>... parameterTypes)

• Method[] getMethods()

Important Methods of java.lang.reflect.Field

- Object get(Object obj)
- String getName()
- Class<?> getType()

Important Methods of java.lang.reflect.Constructor

- int getParameterCount()
- Class<?>[] getParameterTypes()
- T newInstance(Object... initargs)

Important Methods of java.lang.reflect.Method

- String getName()
- int getParameterCount()
- Class<?>[] getParameterTypes()
- Class<?> getReturnType()
- Object invoke(Object obj, Object... args)

Using Default Constructors

- To create Java objects of a certain class, the first step is to retrieve one of the constructors defined in the class
- You need to consult the API Specification

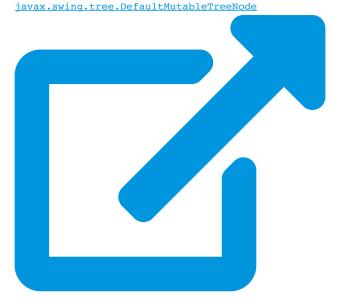


of the class

- The static java.lang.Class.forName method can be called through a java.lang.Class object to retrieve the correct java.lang.Class object of the represented class
- Again, try to maintain the distinction between the type of the object (java.lang.Class) and the class it represents
- We need to pass in the name of the class, or the name of the constructor, into this method
- For illustrative purposes, let us assume that we want to create a

Java tree

The class we want to use is



- In this class, there is a default constructor (a constructor accepting no arguments) which we can call to create a javax.swing.tree.DefaultMutableTreeNode object
- We want to start with the default constructor because we don't want to deal with complication involving parameters required by other constructors
- The first step is to retrieve the java.lang.Class object representing the

```
javax.swing.tree.DefaultMutableTreeNode class:
## step 1: get the Class object
#set( $dmtnClassObj = $_.Class.forName(
  "javax.swing.tree.DefaultMutableTreeNode" ) )
$dmtnClassObj.Class.Name ##=> java.lang.Class,
this is a Class object
$dmtnClassObj.Name ##=>
javax.swing.tree.DefaultMutableTreeNode, the class
represented
```

• The second step is to retrieve the default constructor:

```
## step 2: get the constructor
#set( $dmtnConstructor =
$dmtnClassObj.getConstructor() )
$dmtnConstructor.Class.Name ##=>
java.lang.reflect.Constructor
$dmtnConstructor.Name ##=>
javax.swing.tree.DefaultMutableTreeNode, name of
the constructor
```

• The third step is to use the constructor to create three tree nodes:

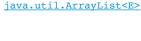
```
## step 3: create three tree nodes
#set( $rootNode = $dmtnConstructor.newInstance() )
$rootNode.Class.Name ##=>
javax.swing.tree.DefaultMutableTreeNode
#set( $leftChild = $dmtnConstructor.newInstance()
)
#set( $rightChild = $dmtnConstructor.newInstance()
```

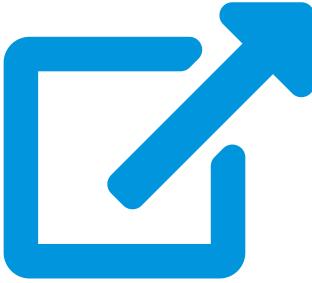
)

• The fourth step is to attach the two child nodes to the root node:

```
## step 4: attach the children to the root
#set( $void = $root.add( $leftChild ) )
#set( $void = $root.add( $rightChild ) )
```

- . Now we have a tree containing three nodes
- To reinforce our understanding, let us create an





object by calling the default constructor of the class:

```
#set( $list = $ .Class.forName(
"java.util.ArrayList"
).getConstructor().newInstance() )
#set( $void = $list.add( 1 ) )
#set( $void = $list.add( 2 ) )
$list ##=> [1, 2]
```

Here I combine all three steps into one line of code. Of course, this line is practically equivalent to set(slist = []). But the point here is to show you how to use the default constructor of any class. If you want to create an object of a certain class, and if you are sure, by consulting the class API, that the class has a default constructor, then replace the String java.util.ArrayList with the class name, and you will have an object of that class.

· Stating the obvious, make sure that the class is not abstract

Retrieving Constructors with Parameters

- The majority of constructors of various classes require parameters
- There are at least two different ways to retrieve a constructor which requires parameters:
- Call java.lang.Class<T>.getConstructors(), which returns all constructors, examine them one by one, and use the one
- Call getConstructor(Class<?>... parameterTypes), which returns the one constructor we want; note that we have to pass in java.lang.Class objects representing the parameters defined with the constructor
- Note that getConstructor(Class<?>... parameterTypes)

requires as parameters either a sequence or an array of java.lang.Class objects

- Note the distinction we made above: what are passed into getConstructor are a set of java.lang.Class objects, each of which representing a class, possibly different than java.lang.Class
- · An example of retrieving a constructor in the first way:

```
## get all constructors
#set( $constructors = $_.Class.forName(
"java.lang.StringBuffer" ).getConstructors() )
$constructors.size() ##=> 4
#foreach( $constructor in $constructors )
    #if( $constructor.getParameterCount() == 1 &&
        $constructor.getParameterTypes()[ 0 ] ==
            $_FieldTool.in( "java.lang.Integer"
).TYPE )
        #set( $myConstructor = $constructor )
        #break
    #end
#end
#set( $myBuffer = $myConstructor.newInstance(
10000 ) )
$myBuffer.Class.Name
```

- In this example, we are trying to retrieve a constructor of the java.lang.StringBuffer class; the constructor in question requires an int argument
- If each constructor of a class requires different number of parameters, then by simply checking the number of parameters of the constructor we want to retrieve (using \$constructor.getParameterCount()), we will be able to pinpoint the one we want
- If the parameter count does not settle on a unique constructor, then we need to look at the types as well
- The java.lang.StringBuffer class has four constructors: the default constructor, and three other constructors, each accepting a single but different parameter:
- StringBuffer()
- StringBuffer(CharSequence seq)
- StringBuffer(int capacity)
- StringBuffer(String str)
- We want to retrieve the one with an int parameter
- int is a primitive type
- To retrieve the java.lang.Class object representing this primitive type, we have to use \$_FieldTool.in(
 "java.lang.Integer").TYPE (a java.lang.Class object)
- Code of this type works for all eight primitive types
 (\$_FieldTool.in("java.lang.Integer").TYPE,
 \$_FieldTool.in("java.lang.Double").TYPE,
 \$_FieldTool.in("java.lang.Boolean").TYPE and so
 on)

- Here we need to check both the parameter count and parameter type to get the constructor we want
- An example of retrieving a constructor in the second way:

```
#set( $myConstructor = $_.Class.forName(
"java.lang.StringBuffer" ).getConstructor(
$_FieldTool.in( "java.lang.Integer" ).TYPE ) )
#set( $myBuffer = $myConstructor.newInstance(
10000 ) )
$myBuffer.Class.Name
```

- Here we pass \$_FieldTool.in("java.lang.Integer").TYPE (again, a java.lang.Class object) directly into the method call of java.lang.Class.getConstructor
- If a constructor requires more than one parameter, then for each parameter, we need to pass in a corresponding java.lang.Class object representing that parameter: ## get the Class objects of parameters #set(\$timeZoneClassObj = \$_.Class.forName("java.util.TimeZone")) #set(\$localeClassObj = \$_.Class.forName("java.util.Locale")) ## retrieve a constructor of java.util.GregorianCalendar that accepts a TimeZone and a Locale #set(\$gCalendarConstructor = \$_.Class.forName("java.util.GregorianCalendar").getConstructor(\$timeZoneClassObj, \$localeClassObj)) \$gCalendarConstructor.Class.Name
- The general procedure to follow, when using getConstructor, is to check the class API, and pass in a whole bunch of java.lang.Class objects, one for each parameter
- . This works if we work with one constructor at a time
- What if we want to have a reusable macro that can be invoked to retrieve any constructor of any class?
- The issue involved here is that a constructor may require zero to N parameters, and if we are to have only one macro, then the java.lang.Class objects to be passed into getConstructor cannot be individual java.lang.Class objects; instead, they have to be passed in as a collection, possibly stored in an java.util.ArrayList object (we are dealing with Velocity here, remember?)
- The problem is that getConstructor does not accept an java.util.ArrayList object when invoked
- We will run into a similar problem when calling java.lang.Class.getMethod, which accepts a method name (i.e., a String) as the first argument, and a collection of java.lang.Class objects as the second argument
- The java.util.ArrayList object storing java.lang.Class objects must be turned into an array of java.lang.Class objects

Reusable getConstructor and getMethod Macros

 Not surprisingly, we need reflection to convert a java.util.ArrayList object into an array

- Note that we are not literally creating an array; instead, we are creating a close approximation of an array that will be accepted by getConstructor and getMethod
- The java.lang.reflect.Array class provides a newInstance(Class<?> componentType, int length) method, which creates a new array with the specified component type and length
- To retrieve the newInstance method using reflection, we have to call getMethod by passing in two java.lang.Class objects, the first one corresponding to the component type (the type of the array), and the second one corresponding to int
- A java.lang.reflect.Method object returned by getMethod can be used to invoke the named method; call the invoke method through this object
- When java.lang.reflect.Method.invoke is called, if the method is an instance method, then the first argument must the the calling object
- On the other hand, if the method is a static method, then the first argument must be null
- That is to say, the number of argument passed into java.lang.reflect.Method.invoke is always one larger than the number of arguments required by the method to be invoked
- Since newInstance is a static method, a null should be passed in as the first argument
- The following snippet shows how to retrieve newInstance and invoke it to create an approximation of an int array:

```
## start with an ArrayList containing
java.lang.Integer objects
#set( $list = [ 1, 2, 3, 4, 5 ] )
## the primitive type int
#set( $intType = $ FieldTool.in(
"java.lang.Integer" ).TYPE )
## create an array of int and of right size
#set( $intArray = $ .Class.forName(
"java.lang.reflect.Array" ).getMethod(
    "newInstance",
    $intType.Class,
    $intType ).invoke(
        null, $intType, $list.size() ) )
## copy data from the ArrayList to the array
#foreach( $item in $list )
    #set( $intArray[ $foreach.index ] = $item )
#end
$intArray.Class.Name
$intArray.size()
#foreach( $item in $intArray )
    $item
#end
```

Because getConstructor and getMethod expects an array of

```
java.lang.Class objects, we need to create an array to store
 java.lang.Class objects
• The component type (the type of the array) we need when
 retrieving newInstance is $_.Class.forName(
 "java.lang.Class" )
• That is to say, we need something like the following:
 ## create an array of Class objects and of right
 size
 #set( $classArray = $_.Class.forName(
 "java.lang.reflect.Array" ).getMethod(
     "newInstance",
     $_.Class.forName( "java.lang.Class" ),
     $_FieldTool.in( "java.lang.Integer" ).TYPE
 ).invoke(
          null, $someType, $someList.size() ) )
• Here is a library macro to do precisely that:
 #macro( chanwConvertArrayListToArray
 $chanwConvertArrayListToArrayList
 $chanwConvertArrayListToArrayType )
     #set( $chanwConvertArrayListToArray = "" )
     #if( $chanwConvertArrayListToArrayList &&
 $chanwConvertArrayListToArrayList.size() > 0 )
          ## determine the type of the array to be
 created
          #if( $chanwConvertArrayListToArrayType &&
 $chanwConvertArrayListToArrayType.Class.Name ==
 "java.lang.Class" )
              #set( $chanwType =
 $chanwConvertArrayListToArrayType )
          #else
              #set( $chanwType =
 $chanwConvertArrayListToArrayList[ 0 ].Class )
 ##
          ## create a java.lang.reflect.Array object
 of the right type and right size
          #set( $chanwConvertArrayListToArray =
 $_.Class.forName( "java.lang.reflect.Array"
 ).getMethod(
              "newInstance",
              $_.Class.forName( "java.lang.Class" ),
 $_FieldTool.in( "java.lang.Integer" ).TYPE
 ).invoke(
              null, $chanwType,
 $chanwConvertArrayListToArrayList.size() )
 ##
          ## copy data over to the array
          #foreach( $num in [
 1..$chanwConvertArrayListToArrayList.size() ] )
              #set( $chanwConvertArrayListToArray[
 $foreach.index ] =
 $chanwConvertArrayListToArrayList[ $foreach.index
 ] )
          #end
```

#end

#end

- The reason why we need the second parameter (\$chanwConvertArrayListToArrayType): to determine the type of the array, we can of course rely on the type of the first element in the list, assuming the type is not a primitive type; if we want a primitive type like \$_FieldTool.in(
 "java.lang.Integer").TYPE, the type must be passed in as the second argument, or else we will need to compute the primitive type from the wrapper type
- With this macro in place, we can create two library macros to retrieve constructors and methods:

#macro(chanwGetConstructor

```
$chanwGetConstructorClassName
 $chanwGetConstructorParamTypeList )
     #set( $chanwGetConstructor = "" )
     #chanwConvertArrayListToArray(
 $chanwGetConstructorParamTypeList )
     #set( $chanwGetConstructor = $_.Class.forName(
 $chanwGetConstructorClassName ).getConstructor(
 $chanwConvertArrayListToArray ) )
 #end
 #macro( chanwGetMethod $chanwGetMethodClassName
 $chanwGetMethodMethodName
 $chanwGetMethodParamTypeList )
     #set( $chanwGetMethod = "" )
     #chanwConvertArrayListToArray(
 $chanwGetMethodParamTypeList )
     #set( $chanwGetMethod = $_.Class.forName(
 $chanwGetMethodClassName ).getMethod(
 $chanwGetMethodMethodName,
 $chanwConvertArrayListToArray ) )
 #end
· Here is some code to test the two macros:
 #import( "site://_brisk/core/library/velocity
 /chanw/chanw-library-import" )
 ## parameters to be passed into the constructor
 #set( $timeZoneLocaleList = [ $_.Class.forName(
 "java.util.TimeZone" ), $_.Class.forName(
 "java.util.Locale" ) ] )
 #chanwGetConstructor(
 "java.util.GregorianCalendar" $timeZoneLocaleList
 $chanwGetConstructor.Class.Name
 ## use the constructor to create a
 GregorianCalendar object
 #set( $gc = $chanwGetConstructor.newInstance(
 $chanwNYTimeZone, $chanwEnUSLocale ) )
 #set( $oneDoubleList = [ $_FieldTool.in(
 "java.lang.Double" ).TYPE ] )
 #set( $twoDoubleList = [ $_FieldTool.in(
```

```
"java.lang.Double" ).TYPE, $_FieldTool.in(
"java.lang.Double" ).TYPE ] )

## test java.lang.Math.pow
#chanwGetMethod( "java.lang.Math" "pow"
$twoDoubleList )
$chanwGetMethod.invoke( null, 3.1, 2 )

## test java.lang.Math.abs
#chanwGetMethod( "java.lang.Math" "abs"
$oneDoubleList )
$chanwGetMethod.invoke( null, -3.1 )
```

Retrieving Class Constants

- We have been using the \$_FieldTool object of type com.hannonhill.cascade.velocity.CascadeFieldTool to retrieve class constants
- The com.hannonhill.cascade.velocity.CascadeFieldTool class does not provide any useful methods
- com.hannonhill.cascade.velocity.CascadeFieldTool is a subclass of org.apache.velocity.tools.generic.FieldTool
- In the org.apache.velocity.tools.generic.FieldTool class, there is an in method that return a org.apache.velocity.tools.generic.FieldTool.FieldToolSub object, which holds a map of all public static field names (i.e., class constants) to values
- The in method has three favors: it can take a java.lang.String argument, a java.lang.Class argument, or an object argument
- Retrieve the value of a name by tagging the name after the map:
 \$_FieldTool.in("java.lang.Math").PI ## a String
 argument
 #set(\$int = 3)
 \$_FieldTool.in(\$int.Class).MAX_VALUE ## a Class
 object argument
 \$_FieldTool.in(\$int).MAX_VALUE ## an
 object argument

Limitations of \$_FieldTool.in

- \$_FieldTool.in can only retrieve constants defined in a class
- Inherited constants may not be retrievable
- To retrieve an inherited constant, we need to know in which class the constant is defined; without this information, the field is not retrievable
- To solve this problem, we can work with the java.lang.Class and java.lang.reflect.Field classes, because the java.lang.Class.getField(String name) method returns a field bearing the name, even if the field is inherited
- There are also constants defined in Cascade classes that are not retrievable using \$ FieldTool.in

Using

#chanwGetConstantValueByClassNameConstantName

· A library macro to retrieve a field value: #macro(chanwGetConstantValueByClassNameConstantName $\verb§chanwGetConstantValueByClassNameConstantNameClassName and \verb§chanwGetConstantNameClassNameConstantNameClassName and \verb§chanwGetConstantNameClassNameConstantNameClassNameConstantNameClassName and \verb§chanwGetConstantNameClassNameConstantNameClassNameConstantNameClassNameConstantNameClassNameConstantNameClassNameConstantNameClassNameConstantNameClassNameConstantNameClassNameConstantNameClassNameConstantNameConstantNameClassNameConstantNameClassNameConstantNameClassNameConstantNameClassNameConstantNameClassNameConstantNameClassNameConstantName$ $\verb§chanwGetConstantValueByClassNameConstantNameConsta$) ## check the class name string #if($\verb!\$chanwGetConstantValueByClassNameConstantNameClassName$ $\verb§chanwGetConstantValueByClassNameConstantNameClassName.Class.Name is a substitute of the property of the pr$!= "java.lang.String" || \$chanwGetConstantValueByClassNameConstantNameClassName == "") #chanwOutputErrorMsg(\$CHANW_LIBRARY_FOLDER_PATH \$chanw_framework_site_name \$CHANW INITIALIZATION FORMAT NAME "A valid class name is required." "kfovm;34go;i") #stop #end ## ## check the constant name string #if($! \verb|\$chanwGetConstantValueByClassNameConstantNameConstantName|$ $\verb§chanwGetConstantValueByClassNameConstantNameConstantName.Class.Name and \verb§chanwGetConstantName(Class.Name(Class.NameConstantName(Class.Name(Class.NameConstantName(Class.Name(Class$!= "java.lang.String" || $\verb§chanwGetConstantValueByClassNameConstantNameConsta$ #chanwOutputErrorMsg(\$CHANW_LIBRARY_FOLDER_PATH \$chanw_framework_site_name \$CHANW_INITIALIZATION_FORMAT_NAME "A valid constant name is required." "lk34uo4jni") #stop #end ## ## get the field value \$chanwGetConstantValueByClassNameConstantName = "") #set(\$chanwClassObj = \$_.Class.forName($\verb§chanwGetConstantValueByClassNameConstantNameClassName$)) #set(\$chanwFieldObj = \$chanwClassObj.getField($\verb§chanwGetConstantValueByClassNameConstantNameConsta$)) \$chanwGetConstantValueByClassNameConstantName = \$chanwFieldObj.get(null))

13 of 18 5/23/19, 3:16 PM

not found

#if(

• To use this macro, use code of the following type:

```
#chanwGetConstantValueByClassNameConstantName('com.hannonhill.cascade.velocity.Ca'MILLIS_PER_YEAR')
$chanwGetConstantValueByClassNameConstantName
```

Creating an Object to Represent Tomorrow

 When working with dates, timestamps, etc., we need to turn to classes like

```
com.hannonhill.cascade.velocity.CascadeComparisonDateTool,
java.util.Date, java.util.Calendar,
java.util.GregorianCalendar, java.time.LocalDate,
and java.text.DateFormat
```

Here is a small script to create an object representing tomorrow:
 #import("site://_brisk/core/library/velocity

```
/chanw/chanw-library-import" )
#chanwLocalDateFromTimestampString(
$_DateTool.SystemTime )
#set( $localDateNow =
$chanwLocalDateFromTimestampString )
#set( $localDateTomorrow = $localDateNow.plusDays(
1 ) )
$localDateTomorrow
```

Calling a static Method

- There are two ways to call a static method of a class:
- Through the Class object directly
- Through the java.lang.reflect.Method.invoke method
- When we call \$_.class.forName, we are calling a static method in the first way
- If we want to call java.lang.Math.abs, we can do this:
 \$_.class.forName("java.lang.Math").abs(-3.2)
- The \$_.class.forName("java.lang.Math") portion of the code returns a java.lang.Class object representing the java.lang.Math class, and through this object, abs can be called
- However, this way of calling static methods can lead to unexpected problems
- Consider the following:

```
#set( $chanwLocalDateNow = $_.Class.forName(
"java.time.LocalDate" ).now() )
$chanwLocalDateNow.Class.Name
#set( $chanwInstantNow = $_.Class.forName(
"java.time.Instant" ).now() )
```

\$chanwInstantNow.Class.Name

- The static now method is defined in both java.time.LocalDate and java.time.Instant
- · However, the output is the following:

```
java.time.LocalDate
java.time.LocalDate
```

That is to say, the java.time.Instant.now method is not called, and we get two java.time.LocalDate objects.

. If we change the order of the two method calls:

```
#set( $chanwInstantNow = $_.Class.forName(
"java.time.Instant" ).now() )
$chanwInstantNow.Class.Name
#set( $chanwLocalDateNow = $_.Class.forName(
"java.time.LocalDate" ).now() )
$chanwLocalDateNow.Class.Name
```

We get two java.time.Instant objects instead.

- Therefore, I would not recommend this way of calling a static method
- Always use getMethod or the library macro chanwGetMethod

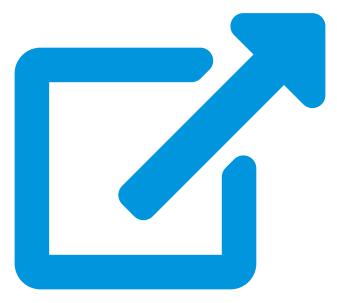
To Parse a String into an int

• Using similar technique, we can invoke

- Note that here we are passing in a java.lang.Class object representing the java.lang.String class into the second call of forName
- Problem: The String to be parsed can come from a field and in fact can be non-numeric; this will cause a NumberFormatException
- There is no try...catch in Velocity; how do we prevent this?

Creating a Calendar of the Current Month

Here is a small script used to generate a calendar of the current month: <u>calendar of current month.vm</u>



Points of interest:

- Constant values like Calendar.MONTH and Calendar.YEAR are used to retrieve the current month and current year
- Calendar.DAY_OF_WEEK is used to retrieve the day of week to help the calculation of number of empty td elements to output
- The modulus operator % is used to align the days properly within the table
- If we want to create a calendar for the entire year, then instead of retrieving the current month, we can loop through [0..11] for the month variable and generate twelve monthly calendars
- If we do not want to use the java.util.GregorianCalendar class (its add method and other related methods are mutable), we can use the java.time.LocalDate class instead

Working with getAttribute and getAttributeValue

- When dealing with an org.jdom.Element object, we may want to retrieve an attribute of an element by calling getAttribute or getAttributeValue
- getAttribute or getAttributeValue can be called with a String parameter (the name of the attribute) when no namespaces are involved
- However, when a namespace is involved, we need to pass in a org.jdom.Namespace object as the second argument
- Here is an example showing how to do that:

```
#*
This format shows how to create a Namespace
object,
used in the method call of getAttribute,
and retrieve data from a org.jdom.Element object.
*#
#import( "site://_brisk/core/library/velocity
/chanw/chanw-initialization" )
#set( $ns = $_.Class.forName( "org.jdom.Namespace")
```

```
).getMethod(
    "getNamespace",
$JAVA_LANG_STRING_CLASS_NAME.Class,
    $JAVA_LANG_STRING_CLASS_NAME.Class ).invoke(
        null, "ss", "http://www.upstate.edu/ss/" )
)
#set( $xml = '<Cell</pre>
xmlns:ss="http://www.upstate.edu/ss/" ' +
    'ss:StyleID="s62"><Data
ss:Type="DateTime">2018-08-14T00:00:00.000</Data>'
    '</Cell>' )
#chanwBuildXMLContentRoot( $xml )
##=> s62
$chanwBuildXMLContentRoot.getAttribute( "StyleID",
$ns ).Value
$chanwBuildXMLContentRoot.getAttributeValue(
"StyleID", $ns )
##=> 2018-08-14T00:00:00.000
$chanwBuildXMLContentRoot.getChild( "Data" ).Value
##=> DateTime
$chanwBuildXMLContentRoot.getChild( "Data"
).getAttributeValue( "Type", $ns )
##=> Type
$_XPathTool.selectSingleNode(
$chanwBuildXMLContentRoot, "Data/@*[local-
name()='Type']" ).Name
##=> DateTime
$_XPathTool.selectSingleNode(
$chanwBuildXMLContentRoot, "Data/@*[local-
name()='Type']" ).Value
##=> 2018-08-14T00:00:00.000
$_XPathTool.selectSingleNode(
$chanwBuildXMLContentRoot, "Data[@*[local-
name()='Type']]" ).Value
##=> 2018-08-14T00:00:00.000
$_XPathTool.selectSingleNode(
$chanwBuildXMLContentRoot,
"Data[@ss:Type='DateTime']" ).Value
```

Creating Global Utility Objects

- Since objects like the org.apache.commons.lang.StringUtils object can be reused in any format, it is better to create them in a library format
- In the library format site://_brisk/core/library/velocity/chanw/chanw-initialization, a set of global utility objects have been created
- These global objects can be used anywhere

- These objects are created by retrieving constructors from various classes
- For more details, see Lesson 3: Numbers, Strings, and Dates

Using Reflection to Display Class Information

- The library format chanw-reflect-utilities provides macros to display information of any class deployed in Cascade
- Invoke #chanwDisplayClassAPI by passing in a class name, a
 java.lang.Class object, or an object of any type:
 #chanwDisplayClassAPI("java.lang.Math", "h3",
 "true") ## pass in a class name
 #chanwDisplayClassAPI(\$_, "h3", "true")
 ## pass in an object
 #chanwDisplayClassAPI(\$_.Class, "h3", "true")
 ## pass in a java.lang.Class object
 #chanwDisplayClassAPI(\$_) ## pass in an object