SingletonAttribute

```
#region license
// Copyright (c) 2005, Sorin Ionescu
// All rights reserved.
//
// Redistribution and use in source and
binary forms, with or without modification,
// are permitted provided that the following
conditions are met:
//
//     * Redistributions of source code must
retain the above copyright notice,
//     this list of conditions and the
following disclaimer.
//     * Redistributions in binary form must
reproduce the above copyright notice,
//     this list of conditions and the
following disclaimer in the documentation
//     and/or other materials provided with
the distribution.
//     * Neither the name of Sorin Ionescu
nor the names of its
//     contributors may be used to endorse
or promote products derived from this
//     software without specific prior
written permission.
//
// THIS SOFTWARE IS PROVIDED BY THE
COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
AND
```

```
// ANY EXPRESS OR IMPLIED WARRANTIES,
INCLUDING, BUT NOT LIMITED TO, THE IMPLIED
// WARRANTIES OF MERCHANTABILITY AND FITNESS
FOR A PARTICULAR PURPOSE ARE
// DISCLAIMED. IN NO EVENT SHALL THE
COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE
// FOR ANY DIRECT, INDIRECT, INCIDENTAL,
SPECIAL, EXEMPLARY, OR CONSEQUENTIAL
// DAMAGES (INCLUDING, BUT NOT LIMITED TO,
PROCUREMENT OF SUBSTITUTE GOODS OR
// SERVICES; LOSS OF USE, DATA, OR PROFITS;
OR BUSINESS INTERRUPTION) HOWEVER
// CAUSED AND ON ANY THEORY OF LIABILITY,
WHETHER IN CONTRACT, STRICT LIABILITY,
// OR TORT (INCLUDING NEGLIGENCE OR
OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
OF
// THIS SOFTWARE, EVEN IF ADVISED OF THE
POSSIBILITY OF SUCH DAMAGE.
#endregion

namespace Boo.Lang.Useful.Attributes

import Boo.Lang.Compiler
import Boo.Lang.Compiler.Ast

class
SingletonAttribute(AbstractAstAttribute):
"""
Implements the singleton pattern for a
class.
```

```
@author Sorin Ionescu
@author Rodrigo B. de Oliveira
"""
 _singletonType as ClassDefinition
 _instance as ReferenceExpression
 _singletonLock as ReferenceExpression

 def constructor():
  pass

 override def Apply(node as Node):
  assert node isa ClassDefinition

  _singletonType = node as ClassDefinition

  MakeClassFinal()
  MakeConstructorsPrivate()
  CreateInstanceField()
  CreateSingletonLockField()
  CreateInstanceProperty()

 private def MakeClassFinal():
  _singletonType.Modifiers |=
TypeMemberModifiers.Final

 private def MakeConstructorsPrivate():
  for member in _singletonType.Members:
   ctor = member as Constructor

   if ctor is not null:
```

```
    ctor.Modifiers |=
TypeMemberModifiers.Private
    ctorFound = true

  if not ctorFound:
   _singletonType.Members.Add(
    Constructor(
     LexicalInfo: LexicalInfo,
     Modifiers:
TypeMemberModifiers.Private))

 private def CreateInstanceField():
  instance = Field(
   LexicalInfo: LexicalInfo,
   Name: "___instance",
   Modifiers: TypeMemberModifiers.Private |
TypeMemberModifiers.Static,
   Type:
SimpleTypeReference(_singletonType.Name))

  _instance =
ReferenceExpression(instance.Name)

  _singletonType.Members.Add(instance)

 private def CreateSingletonLockField():
  singletonLock  = Field(
   LexicalInfo: LexicalInfo,
   Name: "___singletonLock",
   Modifiers: TypeMemberModifiers.Private |
TypeMemberModifiers.Static,
```

```
    Type: SimpleTypeReference("object"),
    Initializer: MethodInvocationExpression(
     ReferenceExpression("object")))

   _singletonLock =
ReferenceExpression(singletonLock.Name)

   _singletonType.Members.Add(singletonLock)

  private def CreateInstanceProperty():
   instanceProperty = Property(
    LexicalInfo: LexicalInfo,
    Name: "Instance",
    Modifiers: TypeMemberModifiers.Public |
TypeMemberModifiers.Static,
    Type:
SimpleTypeReference(_singletonType.Name))

   instanceGetter = instanceProperty.Getter =
Method(
    LexicalInfo,
    Name: "get")

   instanceCreation =
MethodInvocationExpression(
    LexicalInfo: LexicalInfo,
    Target:
AstUtil.CreateReferenceExpression(_singleton
Type.FullName))

   instanceAssignment = BinaryExpression(
```

```
    LexicalInfo,
    BinaryOperatorType.Assign,
    _instance.CloneNode(),
    instanceCreation)

   ifInstanceNullOne = IfStatement(
    LexicalInfo,
    BinaryExpression(
     LexicalInfo,
     BinaryOperatorType.Equality,
     _instance.CloneNode(),
     NullLiteralExpression(LexicalInfo)),
    Block(LexicalInfo), null)

   ifInstanceNullTwo = IfStatement(
    LexicalInfo,
    BinaryExpression(
     LexicalInfo,
     BinaryOperatorType.Equality,
     _instance.CloneNode(),
     NullLiteralExpression(LexicalInfo)),
    Block(LexicalInfo), null)


ifInstanceNullTwo.TrueBlock.Add(instanceAssi
gnment)

   lockMacro = MacroStatement(LexicalInfo:
LexicalInfo, Name: "lock")
   lockMacro.Arguments.Add(_singletonLock)
   lockMacro.Block.Add(ifInstanceNullTwo)
```

```
ifInstanceNullOne.TrueBlock.Add(lockMacro)

instanceGetter.Body.Add(ifInstanceNullOne)
instanceGetter.Body.Add(
 ReturnStatement(
  LexicalInfo: LexicalInfo,
  Expression: _instance.CloneNode()))
```

```
_singletonType.Members.Add(instanceProperty)
```