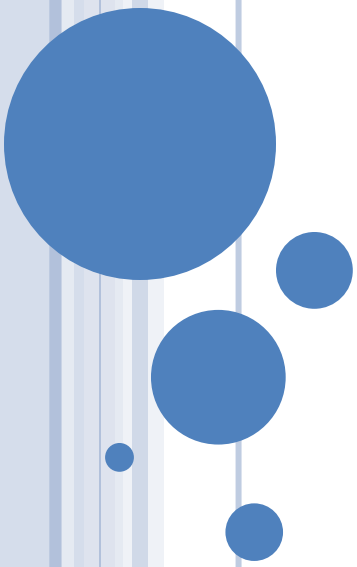


Comp 401 Assertions

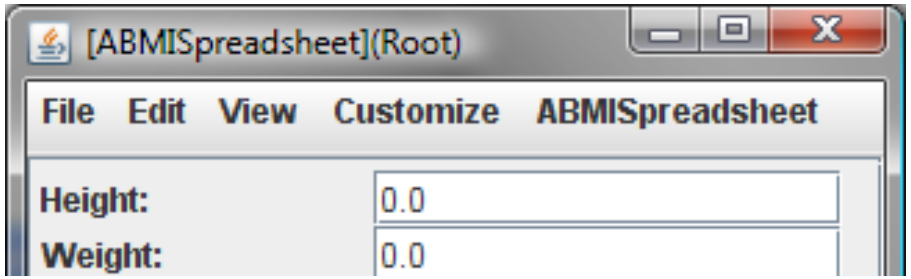
Instructor: Prasun Dewan

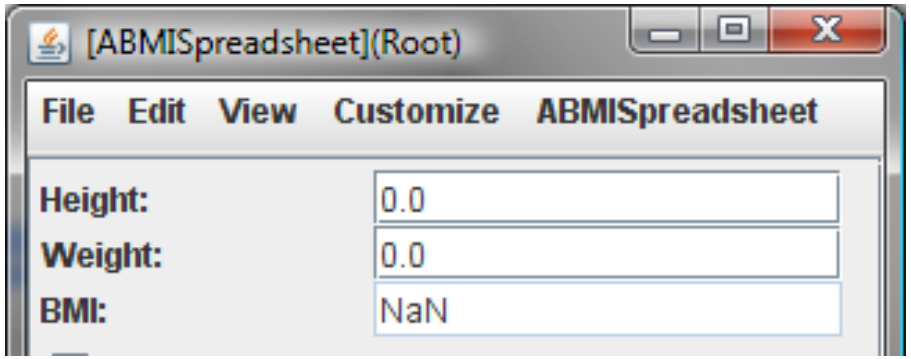
Prerequisite

- Documentation Assertions
- Composite Visitors

.

Preventing Invalid BMI





How Should We Change the Class?

```
public class ABMISpreadsheet {
        double height, weight;

        public ABMISpreadsheet(
                double theInitialHeight, double theInitialWeight) {
                setHeight ( theInitialHeight);
                setWeight( theInitialWeight);
        }

        public double getHeight() { return height; }
        public void setHeight(double newHeight) { height = newHeight; }
        public double getWeight() { return weight; }
        public void setWeight(double newWeight) { weight = newWeight; }
        public double getBMI() { return weight/(height*height); }
}
```

Checking Preconditions

```
public class ABMISpreadsheet {
        double height, weight;

        public ABMISpreadsheet(
                double theInitialHeight, double theInitialWeight) {
                setHeight ( theInitialHeight);
                setWeight( theInitialWeight);
        }
        …
        public boolean preGetBMI() {
                return weight > 0 && height > 0;
        }

        public double getBMI() {
                assert  preGetBMI();
                return weight/(height*height);
        }
}
```

Java Assertions/Pre(Post)Conditions

- **assert** <Boolean Expression>
- **assert** <Boolean Expression>: <Value>
- Statement can be inserted anywhere to state that some condition should be true
- If condition is false, Java throws AssertionError, which may be caught by programmer code.
- If uncaught, depending on which **assert** used:
  - generic message saying assertion failed printed
  - <Value>.toString() printed
- An assertion made at the beginning/end of a statement block (method, loop, if ..) is called its precondition/postcondition

Assertions

- Declare some property of the program
  - Before getBMI() is called, height and weight should be greater than 0

Compile time vs. runtime properties

- Some assertions are language-supported
  - Compile time
    - String s = nextElement()
    - @Override
  - Runtime
    - ((String) nextElement())
    - @util.annotations.ObserverRegisterer(util.annotations.ObserverTypes.VECTOR_LISTENER) addVectorListener(VectorListener)
  - We will consider runtime properties.
  - Casting is application-independent.

Application-independent vs. dependent

- Language can provide us with fixed number of application-independent assertions.
- Cannot handle
  - First character of String is a letter.
  - Letter concept not burnt into language.
    - Class Character defines it
  - Innumerable assertions about letters possible
    - Second elements of string is letter.
    - Third element of string is letter.
- Need mechanism to express arbitrary assertions.
- Originally Java had no assertions.
- In 1.4, assertions were added

Why Language Support

- Can always define a library with assert(<Boolean Expression>) method that throws a special exception denoting assertion error.
- Assertions can be dynamically turned on or off for package or Class
  - java –ea assignment11.MainClass –da bus.uigen…

```
 public void myAssert (boolean boolExp, String message)
throws AssertionError {
        if (!boolExp) throw new AssertionError (message);
    }
```

Error vs. Exception

- Java assertion failure results in AssertionError
- Subclass of Error rather than RunTimeException
- Reasoning:
  - Convention dictates that Exception should be caught
  - Should "discourage programmers from attempting to recover from assertion failures."
    - Might do custom reporting, mail error report etc.
  - AssertionError is a subclass of Throwable and can indeed be caught
  - Decision was controversial

Assertion Uses

- Potentially useful for
  - specification
  - testing
  - formal correctness
  - documentation
  - user-interface automation

Preconditions Conventions

```
public class ABMISpreadsheet {
        double height, weight;

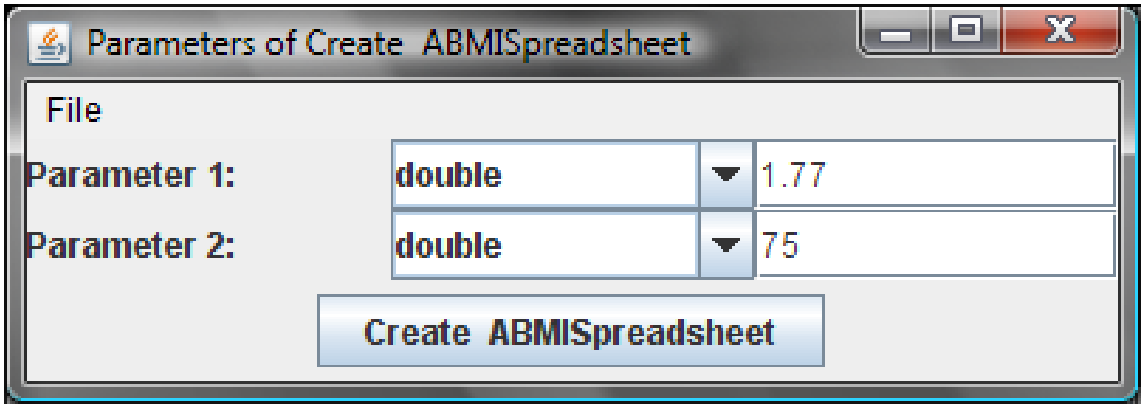
        public ABMISpreadsheet(
                double theInitialHeight, double theInitialWeight) {
                setHeight ( theInitialHeight);
                setWeight( theInitialWeight);
        }
        …
        public boolean preGetBMI() {
                return weight > 0 && height > 0;
        }

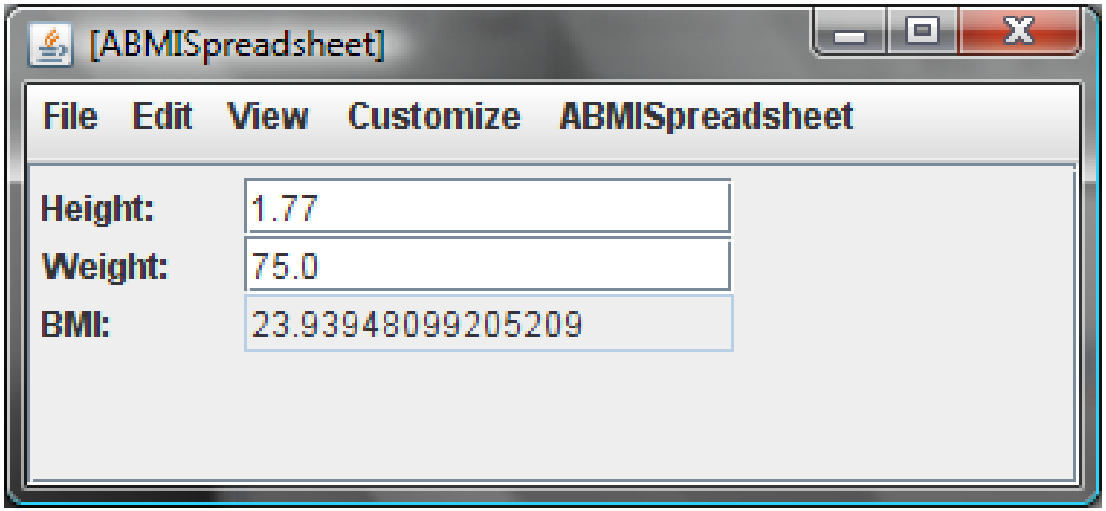
        public double getBMI() {
                assert  (preGetBMI());
                return weight/(height*height);
        }
}
```

Precondition of method M() is preM()

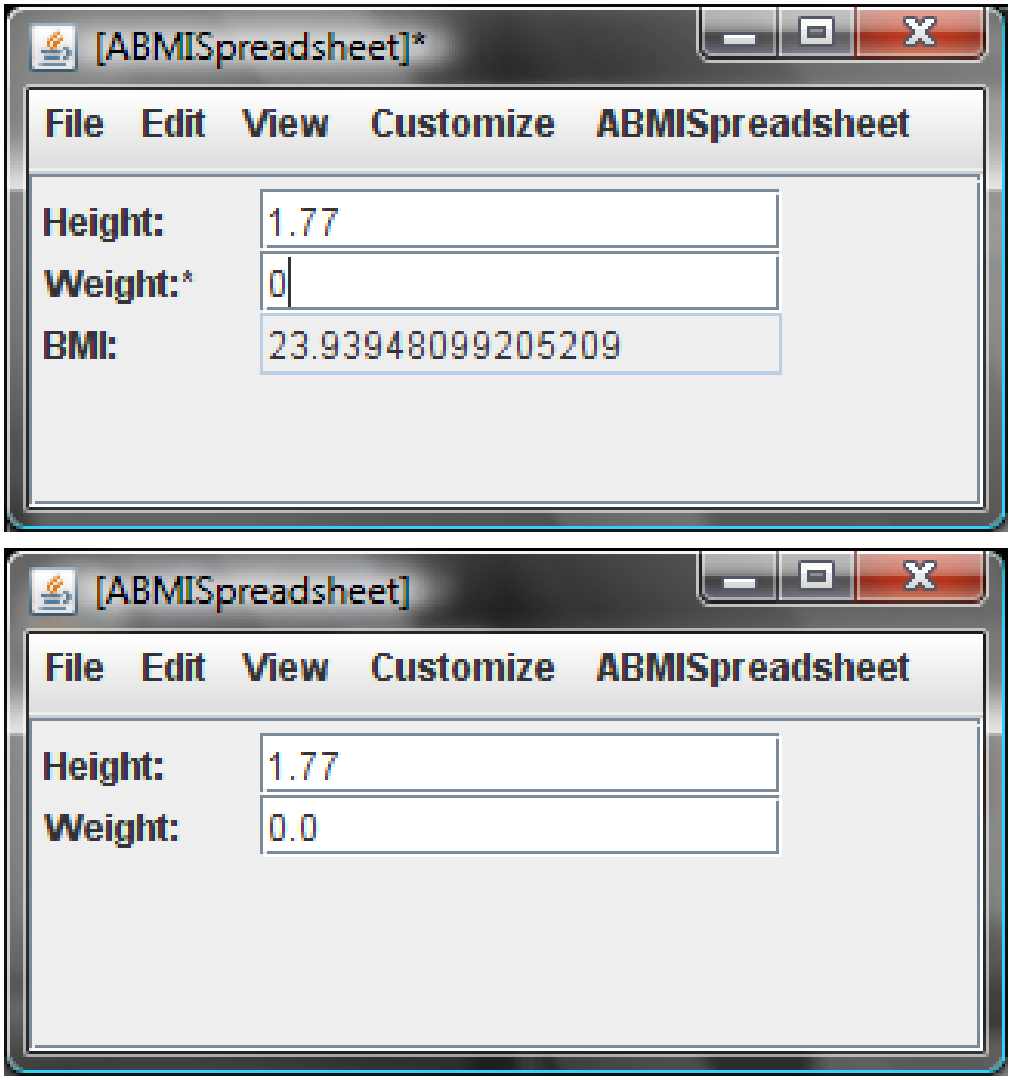
ObjectEditor does not call M() if preM() is false.

ObjectEditor Uses Preconditions

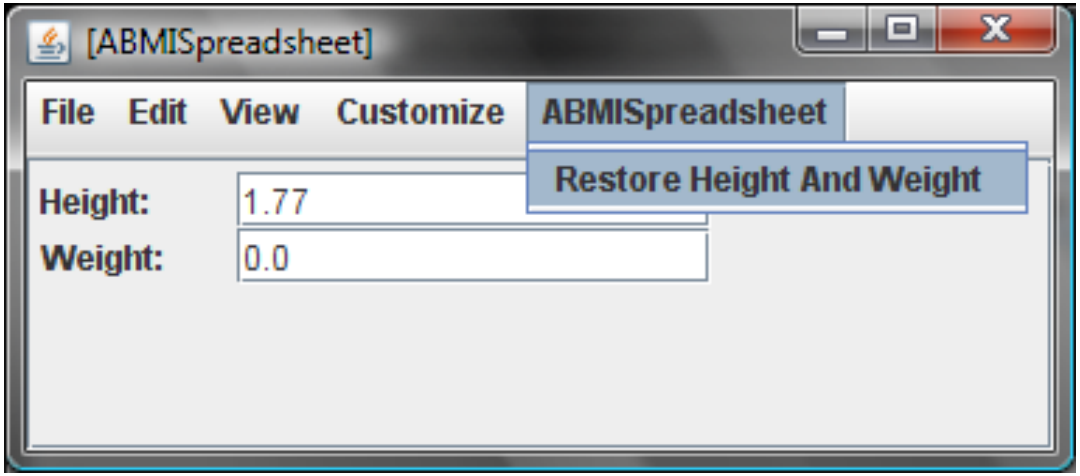


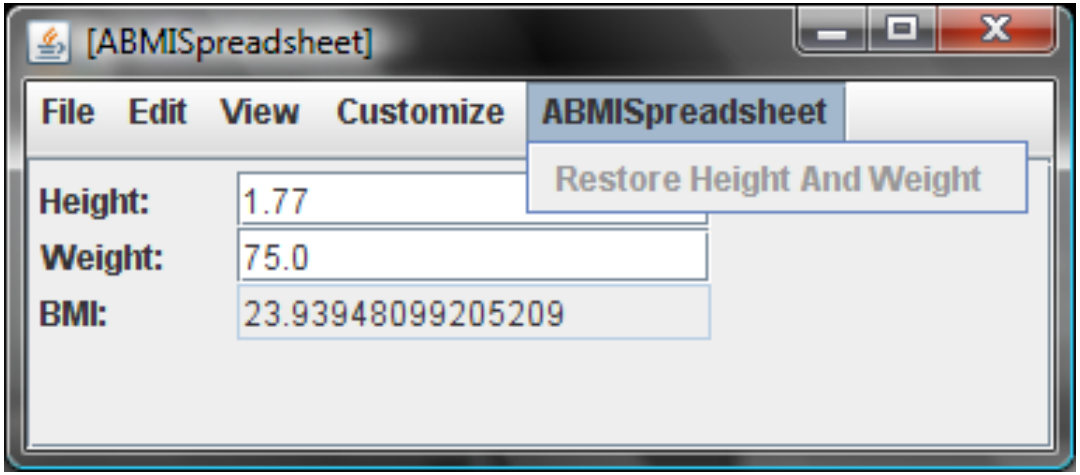


ObjectEditor Uses Precondition



ObjectEditor Uses Precondition





The menu item for a method is disabled when its precondition not met

New Class

```
public class ABMISpreadsheet {
        double height, weight;
        double initialHeight, initialWeight;

        public ABMISpreadsheet(
                double theInitialHeight, double theInitialWeight) {
                setHeight ( theInitialHeight);
                setWeight( theInitialWeight);
                initialHeight = theInitialHeight;
                initialWeight = theInitialWeight;
        }
        …
        public boolean preGetBMI() { return weight > 0 && height > 0; }
        public double getBMI() {
                assert  preGetBMI(); return weight/(height*height); }

        public boolean preRestoreHeightAndWeight() {
                return height != initialHeight || weight != initialWeight; }
        public void restoreHeightAndWeight() {
                assert preRestoreHeightAndWeight();
                height = initialHeight;
                weight = initialWeight;
        }
}
```

Preconditions of Other Methods

```
public class ABMISpreadsheet {
        …
        public double getWeight() {
                return weight;
        }
        public void setWeight(double newWeight) {
                weight = newWeight;
        }
        …
}
```

Preconditions of Other Methods

```
public class ABMISpreadsheet {
        …
        public double preGetWeight() {return weight > 0;}
        public double getWeight() {
                assert preGetWeight();
                return weight;
        }
        public boolean preSetWeight (double newWeight) {
                return newWeight > 0;
        }
        public void setWeight(double newWeight) {
                assert preSetWeight(newWeight);
                weight = newWeight;
        }
        …
}
```

Prevention of getter not needed if setter and constructor prevent assignment of illegal values

Equivalent Class

```
public class ABMISpreadsheet {
        …
        public double getWeight() {
                return weight;
        }
        public boolean preSetWeight (double newWeight) {
                return newWeight > 0;
        }
        public void setWeight(double newWeight) {
                assert preSetWeight(newWeight);
                weight = newWeight;
        }
        …
}
```

Prevention of getter not needed if setter and constructor prevent assignment of illegal values

Precondition Style Rule

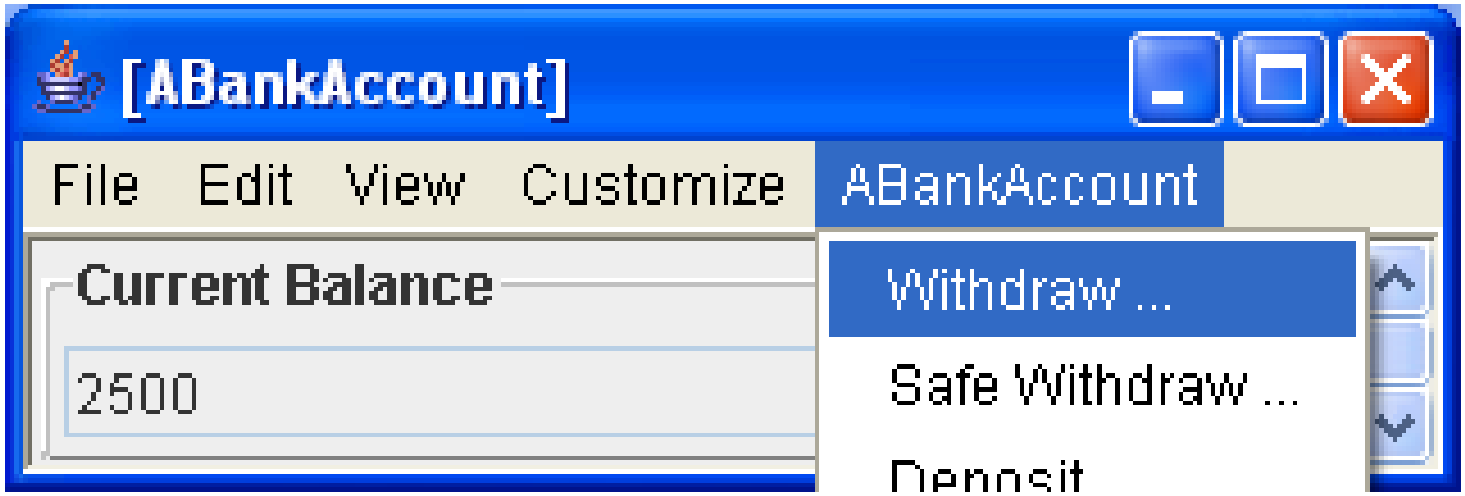
- If there are constraints on the input of a method M(…) that may not be met, write a precondition boolean method, preM(…) for it.
- Call the precondition method in an **assert** statement as the first statement of M(..)
- To keep examples short, preconditions will not be shown in future examples.

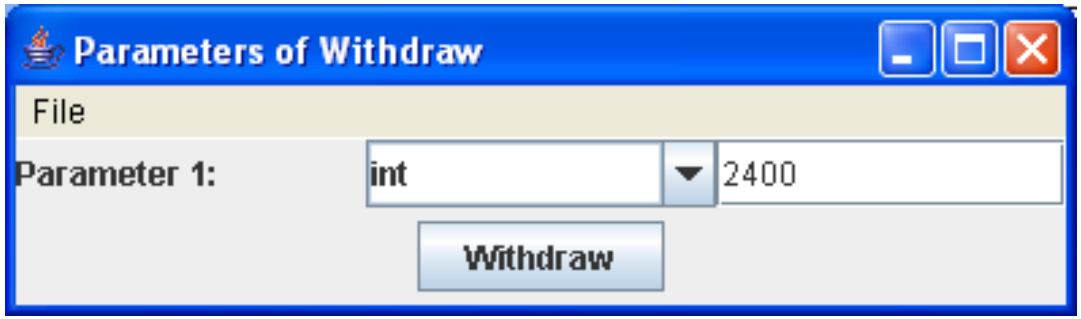
The Importance of Being Earnest

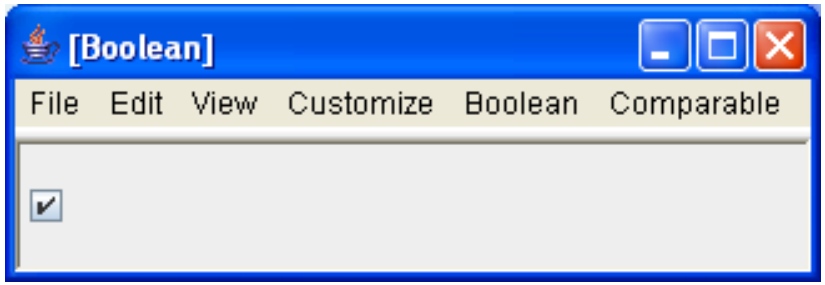
```
public class ABankAccount implements BankAccount {
        int currentBalance = 0;
        public static final int MIN_BALANCE = 100;
        public  ABankAccount (int initialBalance) {
                currentBalance = initialBalance;
        }
        public int getCurrentBalance () {return currentBalance;}
        public void deposit (int amount) {currentBalance += amount;}
        public boolean withdraw (int amount) {
                int minNecessaryBalance = MIN_BALANCE + amount;
                if (minNecessaryBalance  <= currentBalance) {
                        currentBalance -= amount;
                        return true;
                } else return false;
        }

}
```

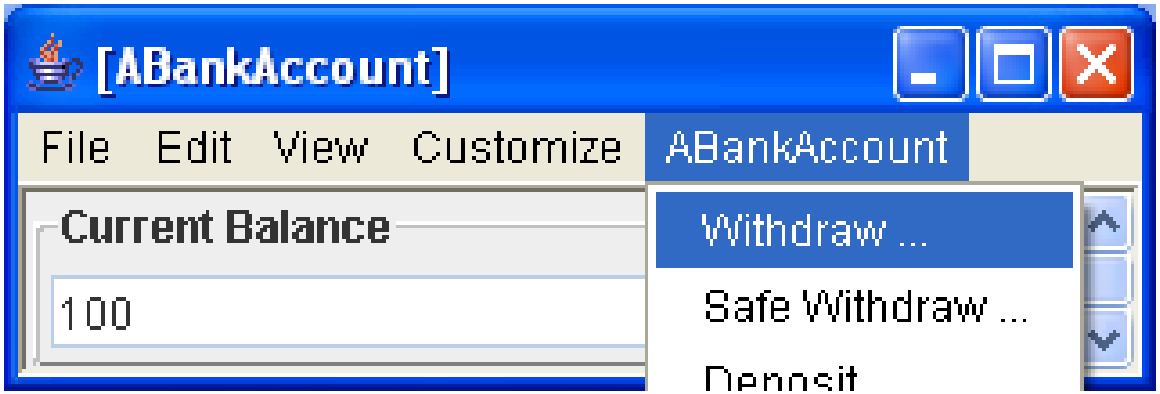
The Importance of Being Earnest

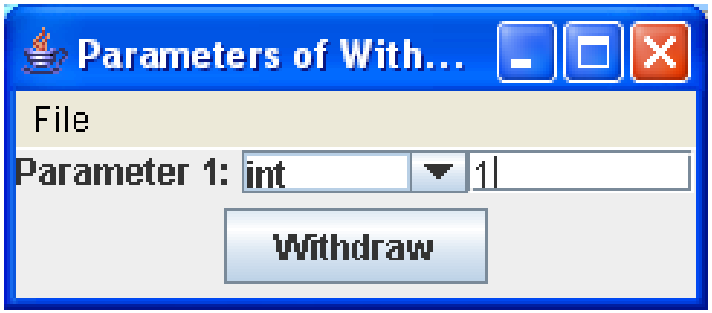


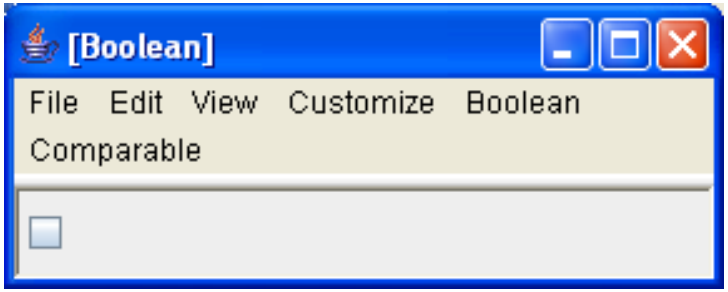




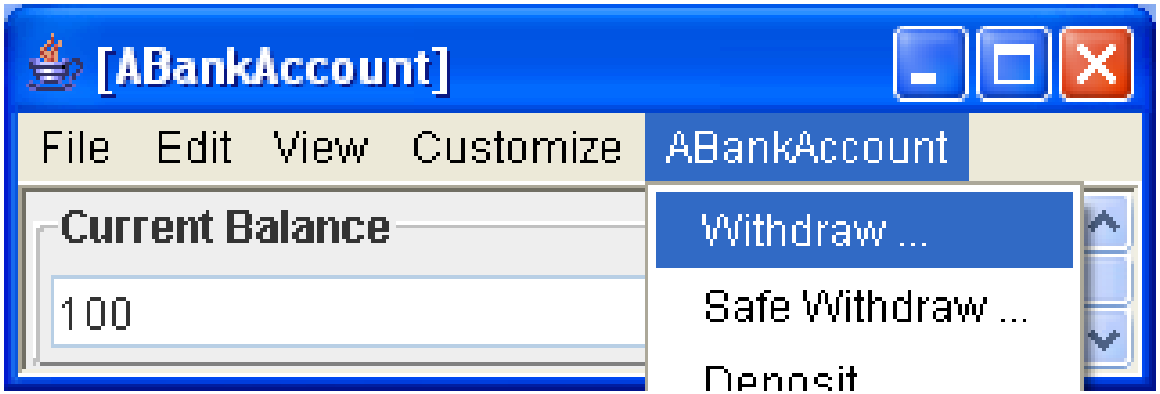
The Importance of Being Earnest

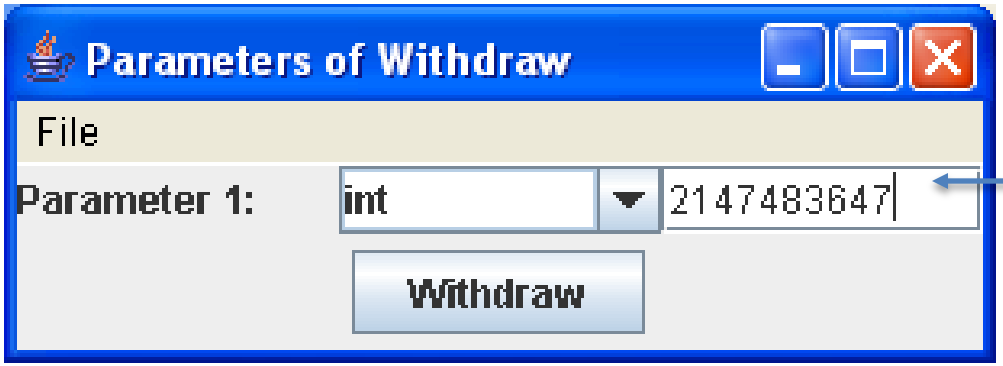




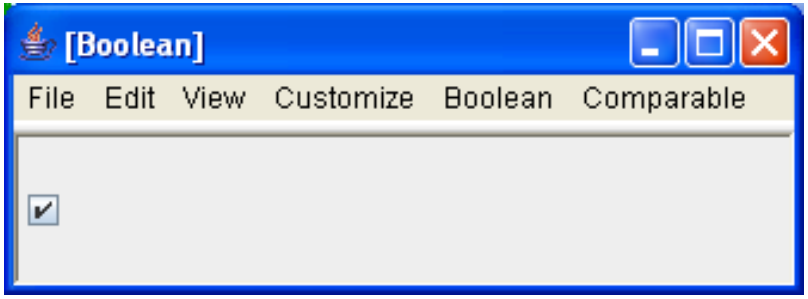


The Importance of Being Earnest

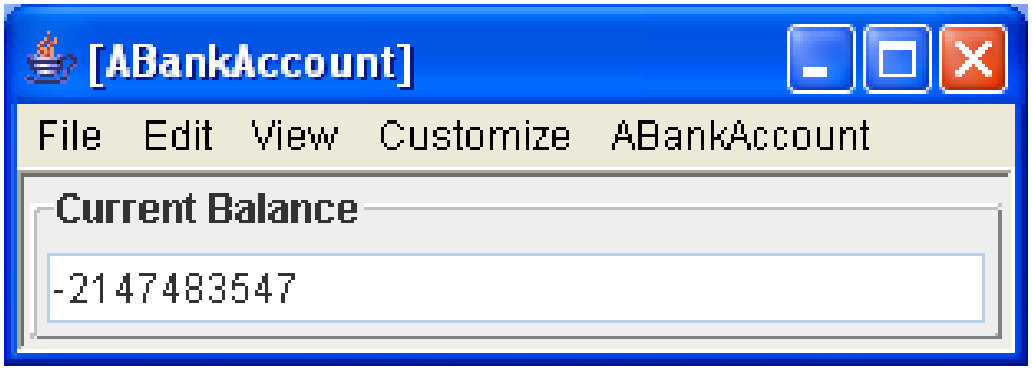




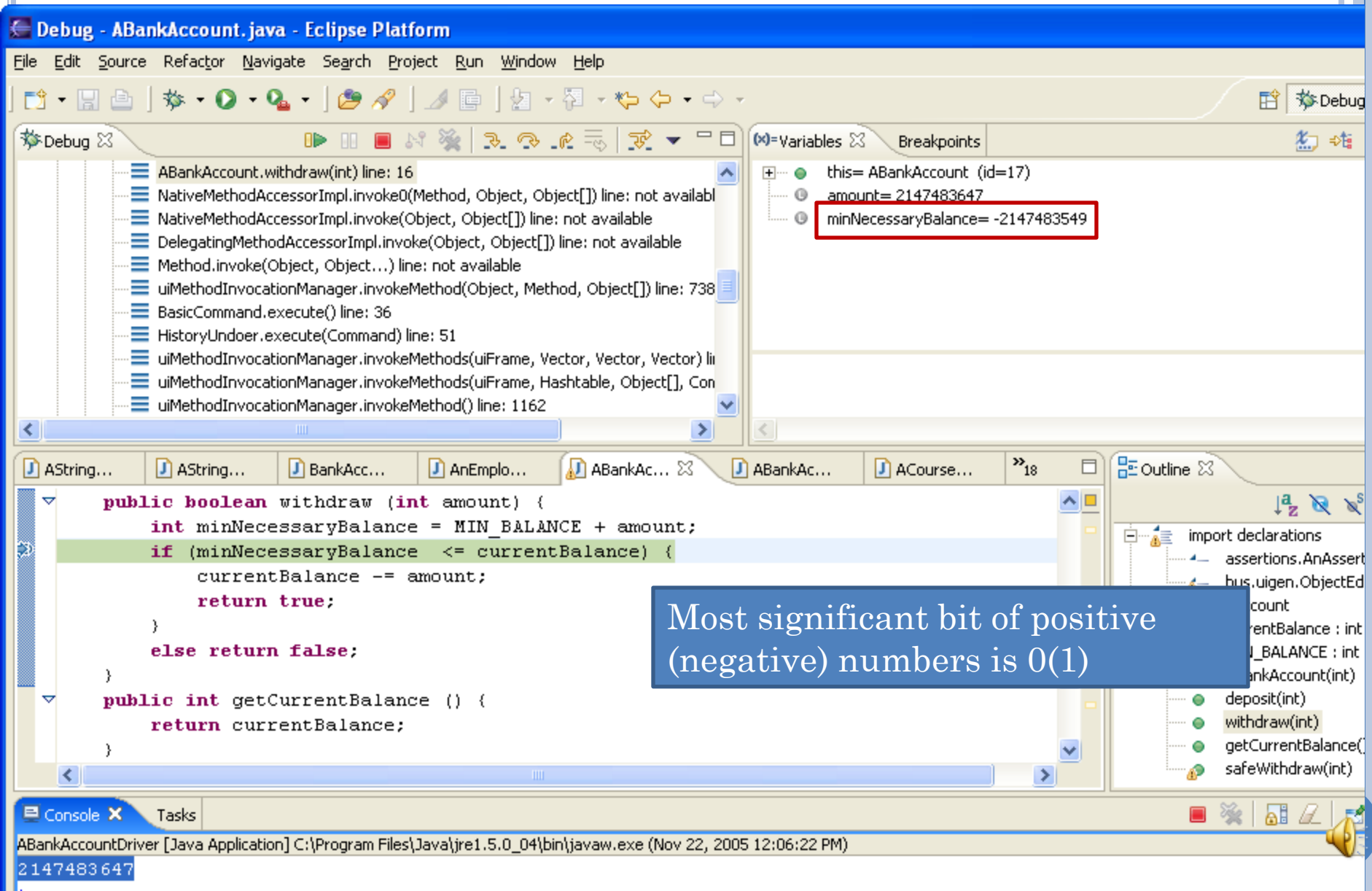
Integer.MAX_INT



The Importance of Being Earnest



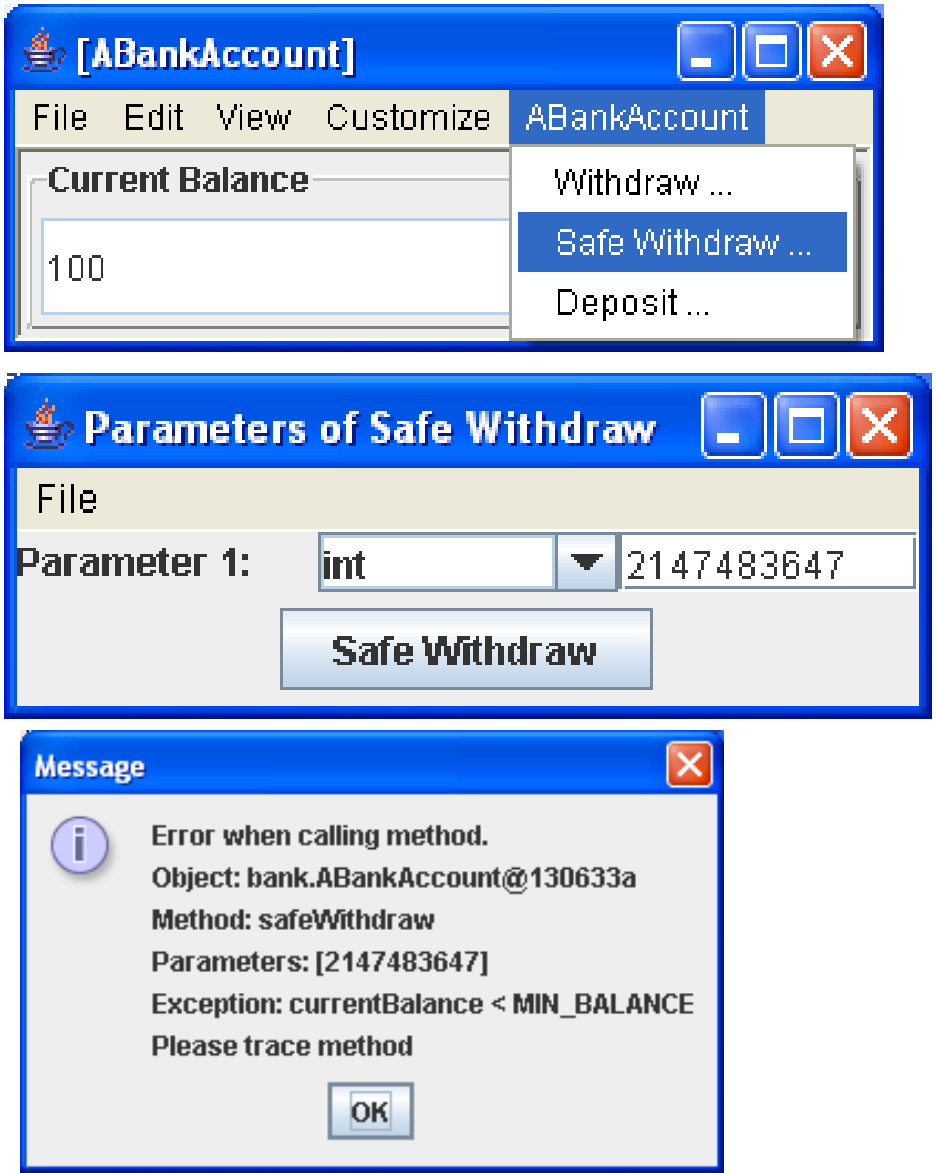
The Importance of Being Earnest



The Importance of Being Earnest

```
        public boolean safeWithdraw (int amount) {
                assert amount > 0: "amount < 0";
                boolean retVal = withdraw(amount);
                assert currentBalance >= MIN_BALANCE: "currentBalance
< MIN_BALANCE");
                return retVal;
        }
```

The Importance of Being Earnest



The Importance of Being Earnest

SecurityTracker.com Archives - Microsoft Internet Explorer **Integer** ...
Microsoft Internet Explorer **Integer Overflow** in Processing Bitmap Files Lets ...
Vendor URL: www.microsoft.com/technet/**security**/ (Links to External Site) ...
www.**security**tracker.com/alerts/2004/Feb/1009067.html - 25k - Cached - Similar pages

Security Issues - Gaim
Description, **Integer overflow** in memory allocation results in heap **overflow**.
By passing the size variable as ~0, **integer overflows** to 0 when 1 is added in ...
gaim.sourceforge.net/**security**/?id=2 - 7k - Cached - Similar pages

Gentoo Linux Documentation -- Samba: **Integer overflow**
Samba: **Integer overflow**. Content:. 1. Gentoo Linux **Security** Advisory, 2.
Impact Information, 3. Resolution Information, 4. References ...
www.gentoo.org/**security**/en/glsa/glsa-200412-13.xml - 9k - Cached - Similar pages

The KOffice Project - XPDF **Integer Overflow** 2
KOffice 1.3 (including betas) to 1.3.5 have an **integer overflow** vulnerability in
KWord's PDF ... References. the corresponding **security** advisiory for KDE. ...
www.koffice.org/**security**/2004_xpdf_**integer_overflow**_2.php - 9k - Cached - Similar pages

Microsoft Windows LoadImage API Function **Integer Overflow** ...
Microsoft Windows is reported prone to a remote **integer overflow** vulnerability.
... Microsoft Upgrade **Security** Update for Windows NT Server 4.0 (KB891711) ...
securityresponse.symantec.com/avcenter/**security**/Content/12095.html - 28k - Cached - Similar pages

CERT Advisory CA-2002-25 **Integer Overflow** In XDR Library
There is an **integer overflow** present in the xdr_array() function ... CERT publications
and other **security** information are available from our web site ...
www.cert.org/advisories/CA-2002-25.html - 26k - Cached - Similar pages

[LSS | **Security** | eXposed by LSS | Detalji]
LSS **Security** Advisory #LSS-2005-01-02. http://**security**.lss.hr. Title:, Apache
mod_auth_radius remote **integer overflow**. Advisory ID:, LSS-2005-01-02 ...
security.lss.hr/en/index.php?page=details&ID=LSS-2005-01-02 - 16k - Cached - Similar pages

Network **Security**, Vulnerability Assessment, Intrusion Prevention
eEye - Network **security** & vulnerability management software including ...
Although the copy length is similarly subject to an **integer overflow**, ...
www.eeye.com/html/research/advisories/AD20051108a.html - 18k - Nov 20, 2005 - Cached - Similar pages

Network **Security**, Vulnerability Assessment, Intrusion Prevention
For the purpose of signature development and further **security** research, a sample

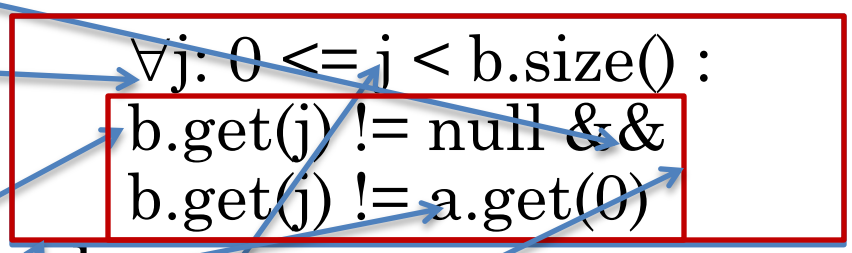
Google for "integer overflow security"

Expressing Assertions

- Natural language
  - No collection element is null.
  - All collection elements are not odd.
  - All collection elements are either odd or positive.
  - Easy to read but ambiguous.
- Programming language
  - Library or language constructs
  - Executable, unambiguous but language-dependent and awkward
    - Useful for debugging
    - Specification cannot be done before language decided.
- Mathematical language
  - Unambiguous, time tested, convenient but not executable
  - ∀j: 0 <= j < b.size() : b.get(j) != null

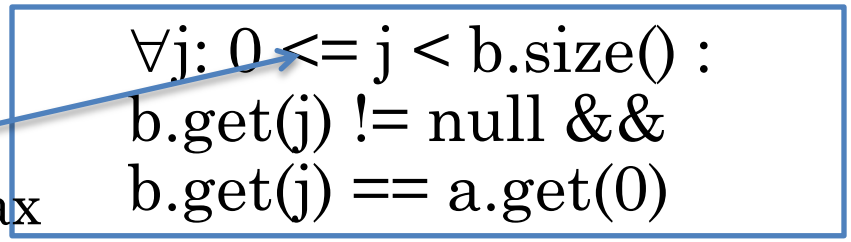
Propositional Calculus

- Logic operators
  - not, and, or
  - We will use Java syntax.
- Quantifiers
  - Universal (∀)
  - Existential (∃)
- Propositional variables
  - Program
  - Others: Recording, Quantifier
- Propositions
  - Boolean expressions involving operators, variables, and quantifiers
- Simple/quantified propositions
  - Do not use/use quantifiers



Propositional Algebra

- Calculus based on algebra
- Algebra defines
  - Arithmetic operations
  - Relations operations
  - We will use Java syntax



Example Propositions

- Simple propositions
  - True
  - False
  - weight > 0
  - (weight > 0) && (height > 0)
- Quantified
  - ∀j: 0 <= j < b.size() : b.get(j) != null
    - All elements of B are not null
  - ∃j: 0 <= j < b.size(): b.get(j) != null
    - At least one element of B is not null.

Quantified Propositions

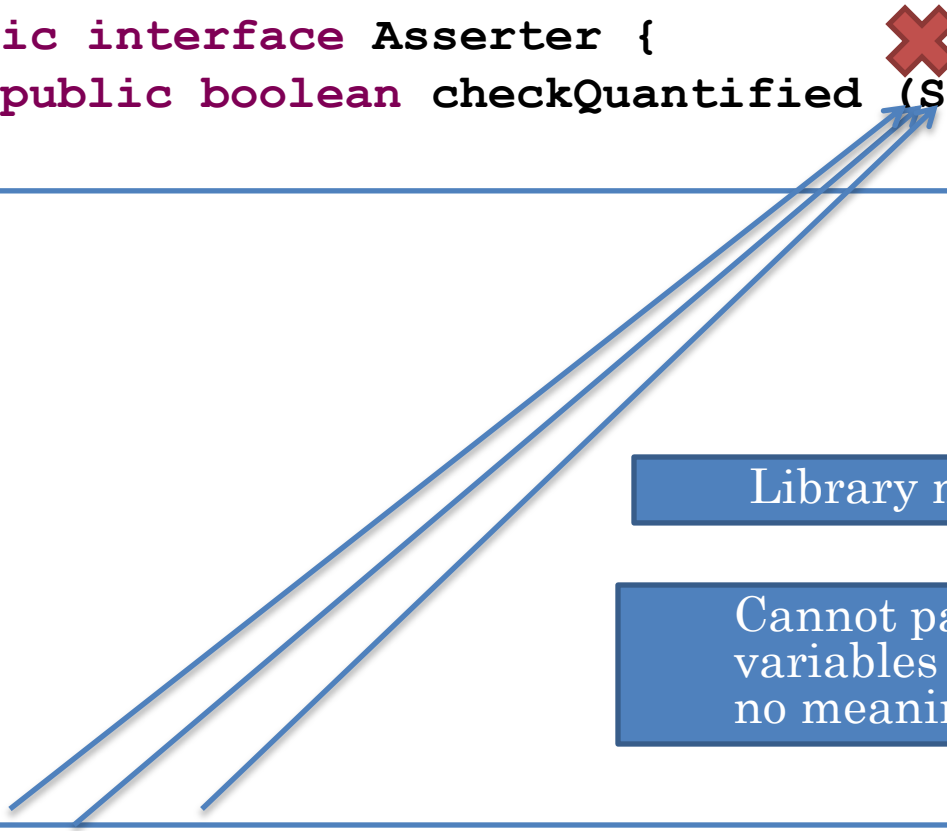
- Quantified
  - ∀j: 0 <= j < b.size(): b.get(j) != null
  - ∃j: 0 <= j < b.size(): b.get(j) != null
- General form:
  - Qx:D(x):P(x)
  - Q is either ∀ or ∃ quantifier
  - X is quantified variable
  - D(x) is domain description
  - P(x) is sub-proposition
- Sub-proposition
  - Simple or quantified proposition in terms of quantifier
- Domain
  - A collection of values used in sub-proposition evaluation
  - b.get(0), … b.get(b.size() – 1)
- Domain description
  - Describes domain using quantified variable

Quantified Assertions

- Syntax
  - Qx:D(x):P(x)
  - ∀j: 0 <= j < b.size(): b.get(j) != null
  - ∃j: 0 <= j < b.size(): b.get(j) != null
- Goal:
  - Write general boolean functions that take as arguments encoding of the elements of domain and return true iff proposition is true

Expressing Quantified Assertions

```
public interface Asserter {
    public boolean checkQuantified (String assertion);
}
```



Library must do parsing.

Cannot pass expression string as variables (such as b) in our scope have no meaning to library

Qx:D(x):P(x)
∀j: 0 <= j < b.size(): b.get(j) != null

assert asserter.checkQuantified("∀j: 0 <= j < b.size(): b.get(j) != null"): "some element of b is null" ;

Separate the Three Components

```
public interface Asserter {
    public boolean checkUniversal (…, …);
    public boolean checkExistential (…, …);
}
```

Separate functions for two quantifiers

One argument for domain and one for predicate

Assume domain is some collection

Qx:D(x):P(x)
∀j: 0 <= j < b.size(): b.get(j) != null

assert asserter.checkUniversal(…,…): “some element of b is null”

List Based Domain

```
public interface Asserter<ElementType> {
    public boolean checkUniversal (List<ElementType>
domain, …);
    public boolean checkExistential (List<ElementType>
domain, …);
}
```

Assume domain implements List

Want to support histories, sets, databases , streams …

Qx:D(x):P(x)
∀j: 0 <= j < b.size(): b.get(j) != null

assert asserter.checkUniversal(b, ….): “some element of b is null” ;

Iterating the Domain

```
public interface Asserter<ElementType> {
    public boolean checkUniversal (Iterator<ElementType>
domain, …);
    public boolean checkExistential (Iterator<ElementType>
domain, …);
}
```

Iterator describes any sequence of ordered/unordered items.

Qx:D(x):P(x)
∀j: 0 <= j < b.size(): b.get(j) != null

assert asserter.checkUniversal(b.iterator(), ….): "some element of b is null")

Asserter Class

```
package util.assertions;
import java.util.Iterator;
public class AnAsserter<ElementType> implements
Asserter<ElementType> {
    public boolean checkUniversal (Iterator<ElementType>
elements, …) {
        while (elements.hasNext())
            if (!…) return false;
        return true;
    }
    public void checkExistential (Iterator<ElementType>
elements, …) {
        while (elements.hasNext())
            if (…) return true;
        return false;
    }
}
```

Describing the Domain Using Method Parameters

```
 public boolean checkUniversal (Iterator<ElementType>
elements, (<ElementType→ boolean)elementChecker ) {
        while (elements.hasNext())
              if (!elementChecker(elements.next())) return
false;
         return true;

}
```

Method must be invoked on some object

Passing method and object separately raises typing issues

```
public boolean nonNullChecker(Object element) {
        return element != null;
}
```

assert asserter.checkUniversal(b.iterator(), nonNullChecker, "some element of b is null")

Describing the Domain Using Method Parameters

```
 public boolean checkUniversal (Iterator<ElementType>
elements, ElementChecker elementChecker ) {
        while (elements.hasNext())
              if (!elementChecker.check(elements.next()))
return false;
         return true;

}
```

Pass an object with a check method that takes argument of type <ElementType>

Qx:D(x):P(x)
∀j: 0 <= j < b.size(): b.get(j) != null

assert asserter.checkUniversal(b.iterator(), **new** ANonNullChecker())): "some element of b is null" ;

Subproposition Visitor Objects

```
package util.assertions;
public interface ElementChecker<ElementType> {
   public boolean check (ElementType element);
}
```

```
import util.assertions.ElementChecker;
public class ANonNullChecker implemen
ElementChecker<Object> {
  public boolean check(Object element) {
       return element != null;
 }}
```

All subproposition objects implement same interface

The check method of a subproposition object visits each

```
 public boolean checkUniversal (Iterator<ElementType>
elements, ElementChecker elementChecker ) {
       while (elements.hasNext())
            if (!elementChecker.check(elements.next()))
return false;
        return true;
}
```

assert asserter.checkUniversal(b.iterator(), **new** ANonNullChecker())): “some element of b is null” ;

Accessing External Vars

```
package util.assertions;
public interface ElementChecker<ElementType> {
   public boolean check (ElementType element);
}
```

```
import util.assertions.ElementChecker;
public class ANonNullChecker implements
ElementChecker<Object> {
  public boolean check(Object element) {
       return element != null;
 }}
```

∀j: 0 <= j < b.size(): b.get(j) != a.get(0)

assert asserter.checkUniversal(b.iterator(), **new** ANonNullChecker())): “some element of b is null” ;

Subproposition accessing vars other than domain elements

```
import util.assertions.ElementChecker;
public class AnInequalityChecker implements ElementChecker<String> {
        Strring testObject;
        public AnInequalityChecker(String theTestObject) {
                testObject = theTestObject;
        }
        public boolean check(String element) {
                return !element.equals(testObject);
        }
}
```

Each external var becomes constructor parameter and checker instance variable

∀j: 0 <= j < b.size(): b.get(j) != a.get(0)

assert asserter.checkUniversal(b.iterator(), **new** AnInquality Checker(a.get(0)): “some element of b == a.get(0)” ;

ActionObject

Action Object = Embedded Opertation

execute (targetObject, params)

getWeightMethod.invoke (bmi, nullParams);

Provides an execute method to perform some embedded operation.

The execute operation takes the object on which the embedded operation is to be invoked and an array of parameters of the target method.

Command Object

Command Object = Embedded Operation + Target + Parameters

execute ()

Constructor (targetObject, params)

```
setWeightCommand.execute();
```

Provides a execute operation to perform some embedded operation.

The execute operation takes no arguments.

Constructor takes parameters of operation as arguments.

Action is an operation that can be invoked on many different arguments

A command is a specific action invocation.

Visitor Object

execute (targetObject)

Constructor (params)

Visitor Object = Embedded Operation + Parameters

```
elementChecker.check(elements.next())
```

```
public boolean check(String element) {
  return !element.equals(testObject);
}
```

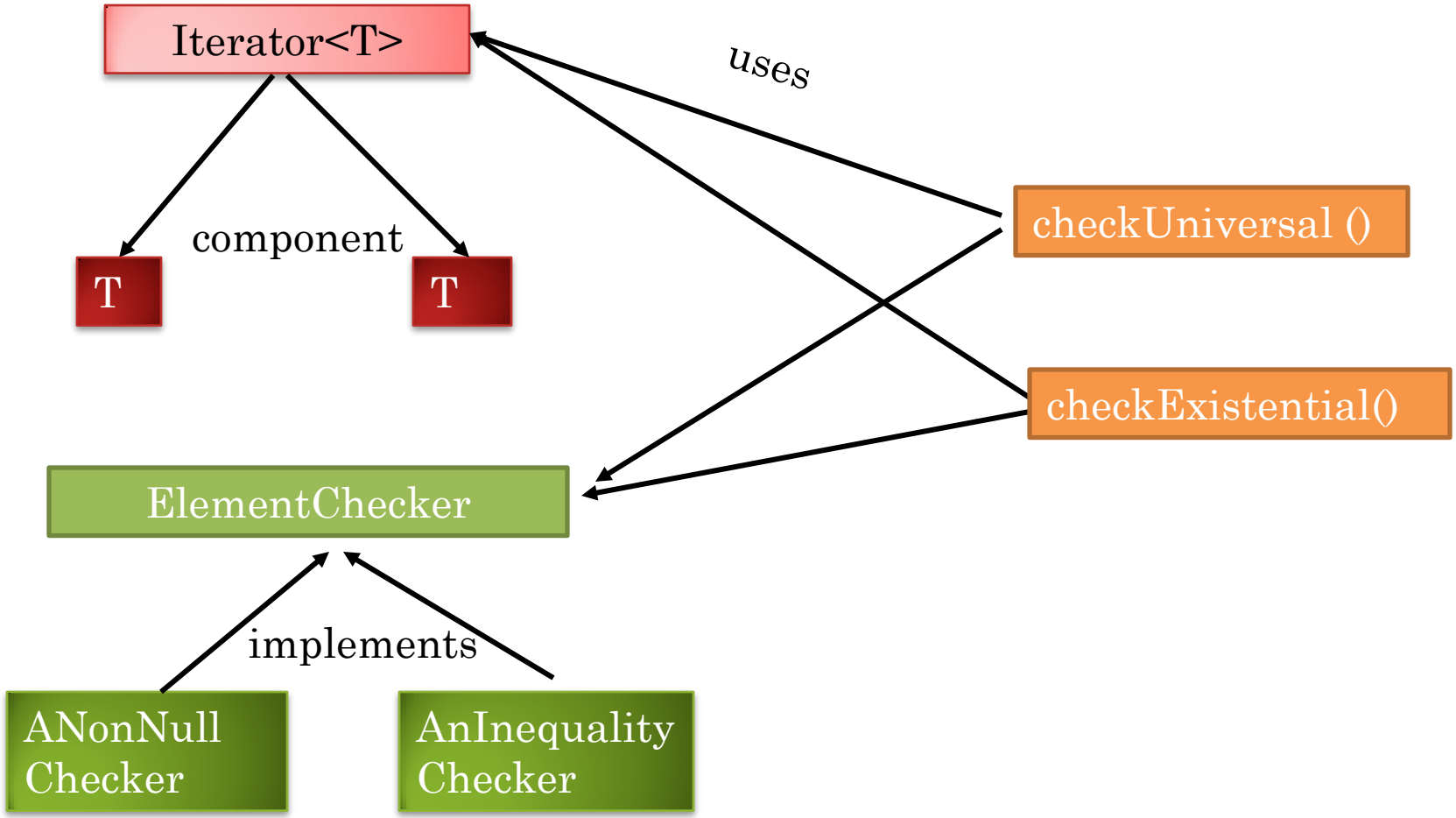
Constructor in (a) command takes target and params, (b) action takes no params, and (c) visitor takes params as arguments.

Provides a execute operation to perform some embedded operation.

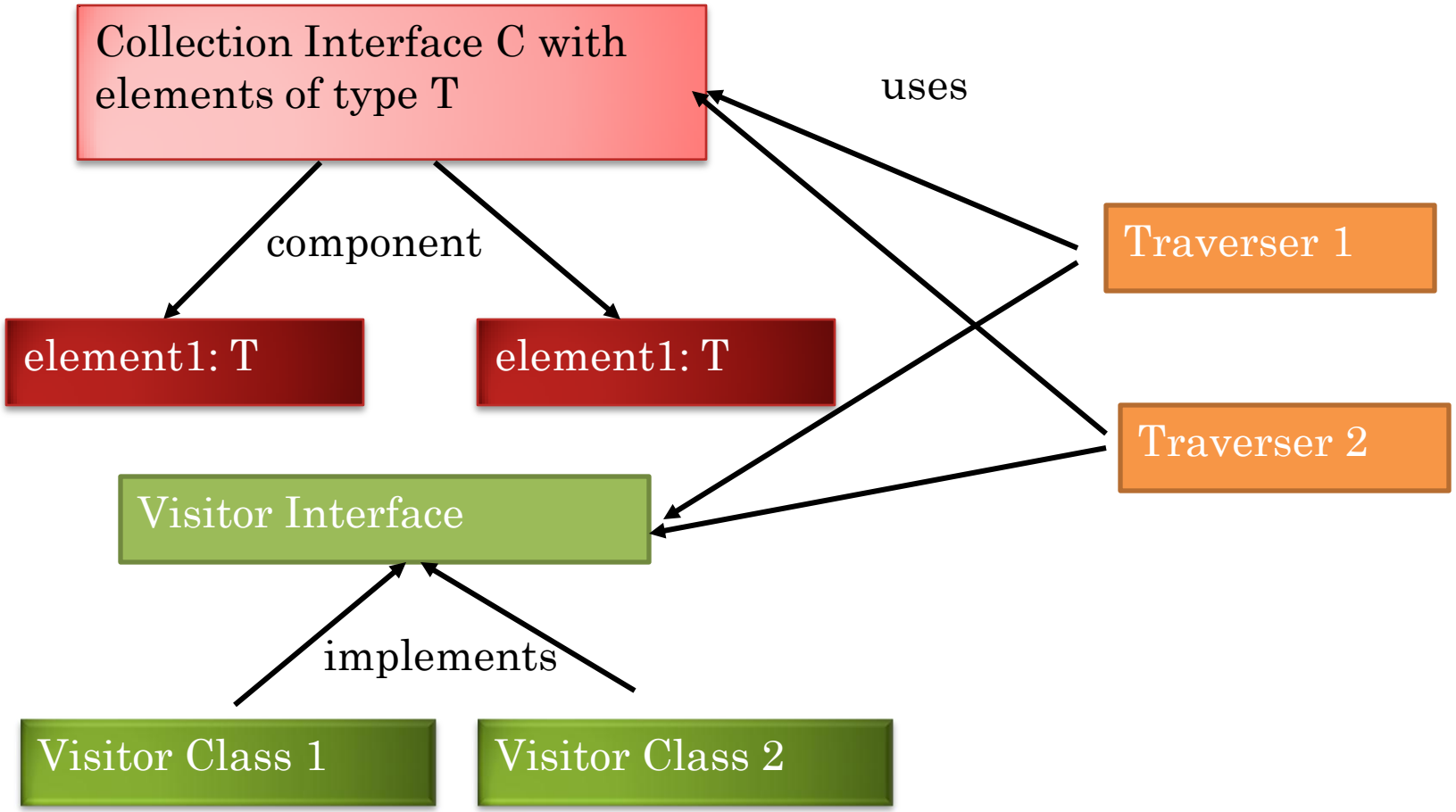
Execute in (a) command takes no params, (b) action takes target object and params, (c) visitor takes target object

The execute operation takes target object as argument

Example of Visitor Pattern



Visitor Pattern



Everyday Visitor Objects

- Compiler visitors for:
  - Formatting program elements
  - Refactoring program elements
  - Compiling program elements.
- ObjectEditor visitors for:
  - Attaching widgets to object components.
  - Registering listeners of object components.
  - Printing a textual representation of object components.

Visitor Pattern

- Some collection C of elements of type T
- Visitor interface
  - **public interface** ElementChecker { **public boolean** visit (T element);}
  - **public interface** V {**public** T2 m (T p);}
- One or more traverser methods that use collection and visitor interface to pass one or more collection elements to the method.
  - **public static boolean** checkUniversal (Iterator<ElementChecker> domain, ElementChecker subProposition) {…subProposition.visit(domain.next ());
  - traverser1 (C c, V v) { …v.m(element of C)…}
- Implementation of interface whose constructors take as arguments external variables that need to be accessed by the visitor method

```
public class AnInequalityChecker implements ElementChecker {
    Object testObject;
    public AnInequalityChecker(Object theTestObject) {…}
    public boolean visit(Object element) {…};
}
public class AV1 implements V {
    public AV1 (T1 p1, … Tn pN) { …}
    public T2 m (T p) { … }
}
```

- Client passes traverser visitor implementation and collection
  - asserter.checkUniversalb.iterate(), **new** AnInequalityChecker(a.get(0)));
  - traverser1(c, **new** AV1(a1,.. aN));

Asserting false

```
switch c  {
      case ‘a’: …
      case ‘b’: …
      default: assert false
}
```

Unreachable statement

Nested Assertions

```
public class AListChecker implements ElementChecker<List> {
    public boolean check(List element) {
        Iterator children = element.iterate();
        return asserter.checkUniversal(children, new
ANonNullChecker()));
    }
}
```

∀j: 0 <= j < b.size(): ∀k: 0 <= k < b.get(j).size(): b.get(j).get(k) != null

```
assert asserter.checkUniversal(b.iterator(), new AListChecker
(),  “some nested  element of b is null)
```

Actual Library

```
package util.assertions;
import java.util.Iterator;
public class AnAsserter<ElementType> implements
Asserter<ElementType> {
    public void assertUniversal (Iterator<ElementType> elements,
ElementChecker elementChecker, String message) {
        while (elements.hasNext())
            if (!elementChecker.check(elements.next())) throw
new AssertionError (message);
    }
    public void assertExistential (Iterator<ElementType>
elements, ElementChecker elementChecker, String message) {
        while (elements.hasNext())
            if (elementChecker.check(elements.next())) return;
        throw new AssertionError (message);
    }
}
```

asserter.assertUniversal(b.iterator(), nonNullChecker, "some element of b is null");

Fails regardless of Java options

Java Assertions

- **assert** <Boolean Expression>
- **assert** <Boolean Expression>: <Value>
- Statement can be inserted anywhere to state that some condition should be true
- If condition is false, Java throws AssertionError, which may be caught by programmer code.
- If uncaught, generic message saying assertion failed is printed.
- An assertion made at the beginning/end of a statement block (method, loop, if ..) is called its precondition/postcondition

Why Language Support

- Can always define a library with assert(<Boolean Expression>) method that throws a special exception denoting assertion error.
- Assertions can be dynamically turned on or off for package or Class
  - java –ea assignment11.MainClass –da bus.uigen…

```
 public void myAssert (boolean boolExp, String message)
throws AssertionError {
        if (boolExp) throw new AssertionError (message);
    }
```

Error vs. Exception

- Java assertion failure results in AssertionError
- Subclass of Error rather than RunTimeException
- Reasoning:
  - Convention dictates that Exception should be caught
  - Should "discourage programmers from attempting to recover from assertion failures."
    - Might do custom reporting, mail error report etc.
  - AssertionError is a subclass of Throwable and can indeed be caught
  - Decision was controversial

The Importance of Being Earnest

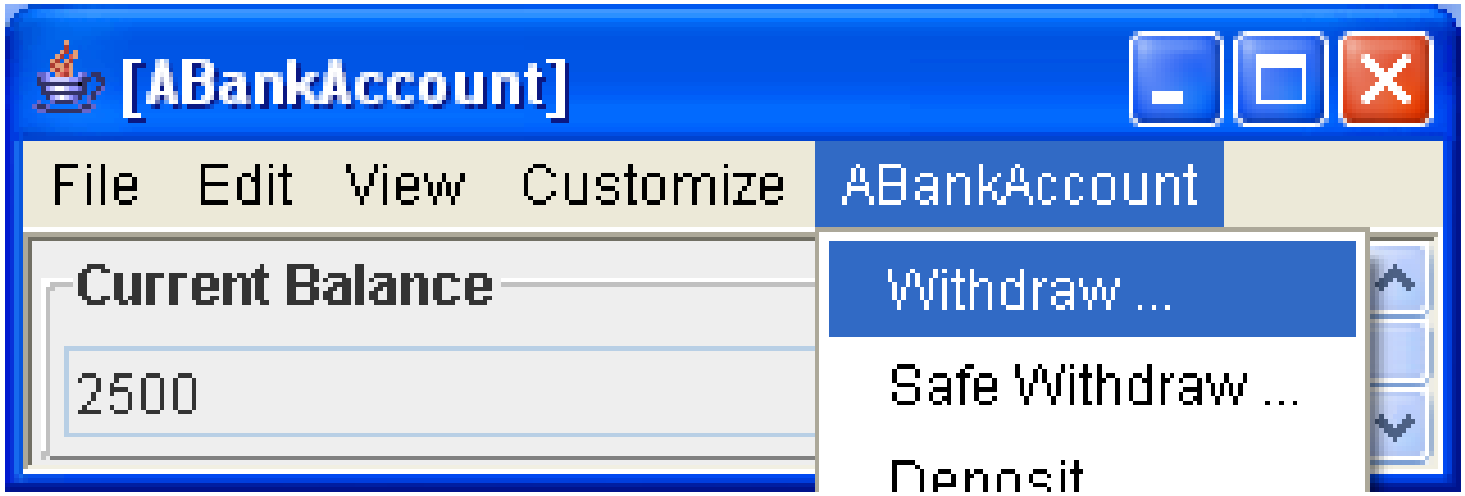
```
public class ABankAccount implements BankAccount {
        int currentBalance = 0;
        public static final int MIN_BALANCE = 100;
        public  ABankAccount (int initialBalance) {
                currentBalance = initialBalance;
        }
        public int getCurrentBalance () {return currentBalance;}
        public void deposit (int amount) {currentBalance += amount;}
        public boolean withdraw (int amount) {
                int minNecessaryBalance = MIN_BALANCE + amount;
                if (minNecessaryBalance  <= currentBalance) {
                        currentBalance -= amount;
                        return true;
                } else return false;
        }

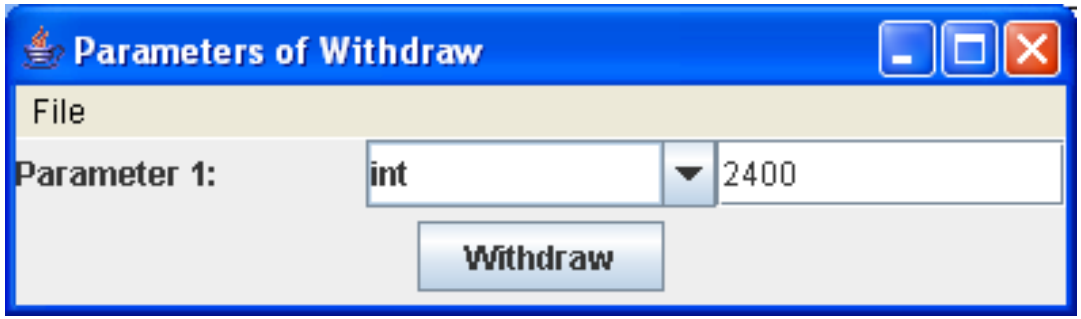
}
```

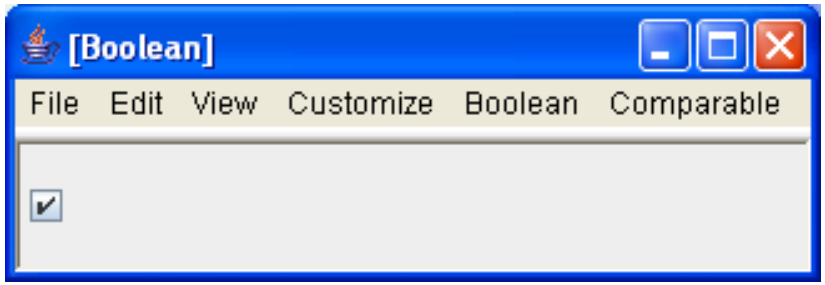
The Importance of Being Earnest

```
        public boolean safeWithdraw (int amount) {
                assert amount > 0: "amount < 0";
                boolean retVal = withdraw(amount);
                assert currentBalance >= MIN_BALANCE: "currentBalance
< MIN_BALANCE");
                return retVal;
        }
```

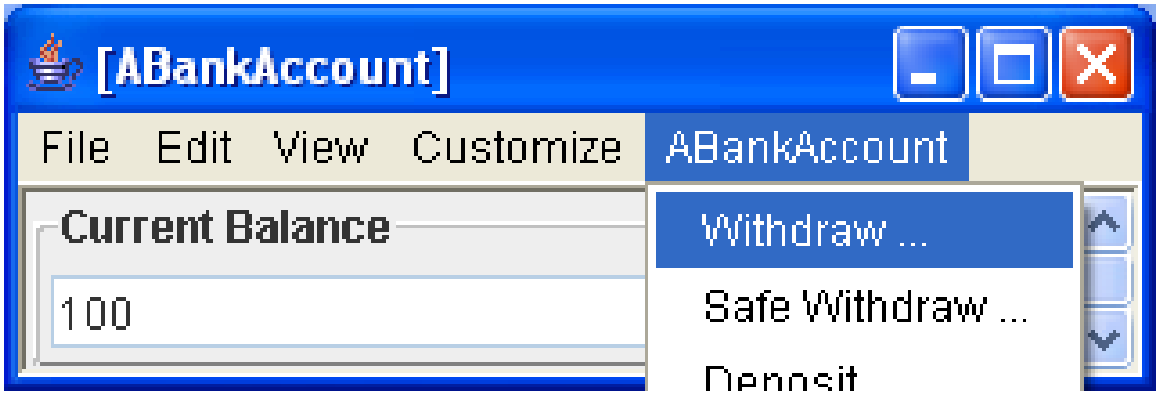
The Importance of Being Earnest

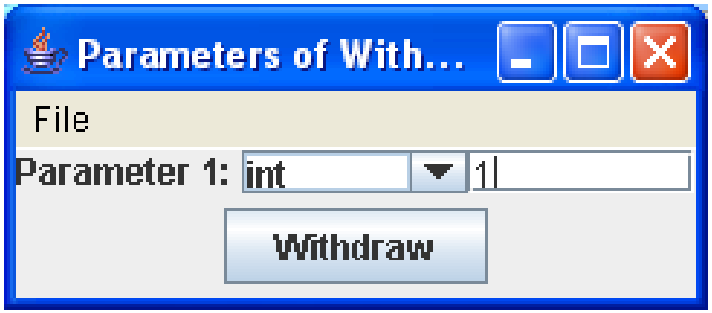


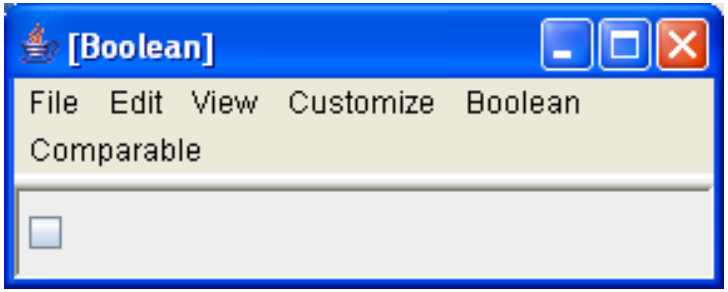




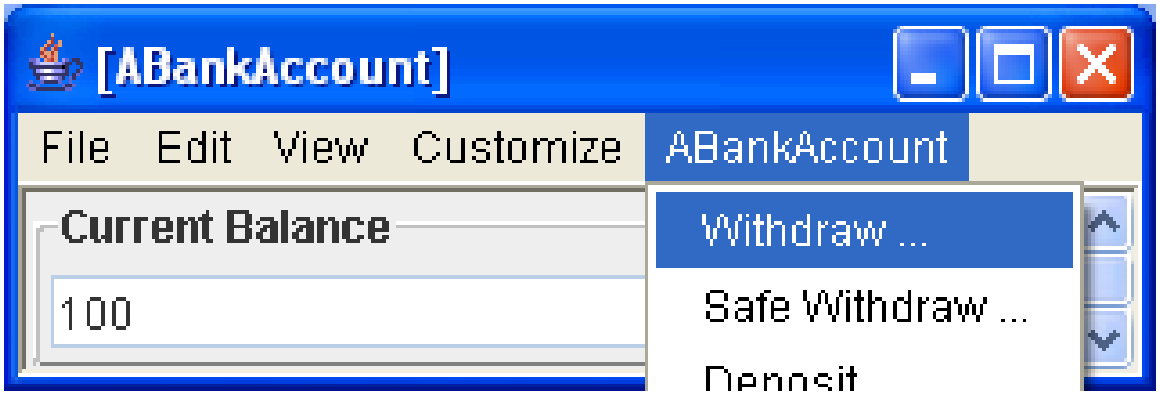
The Importance of Being Earnest

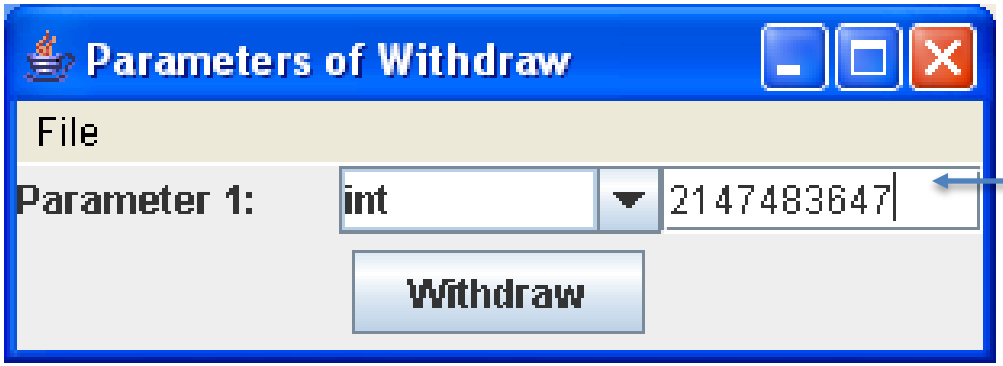




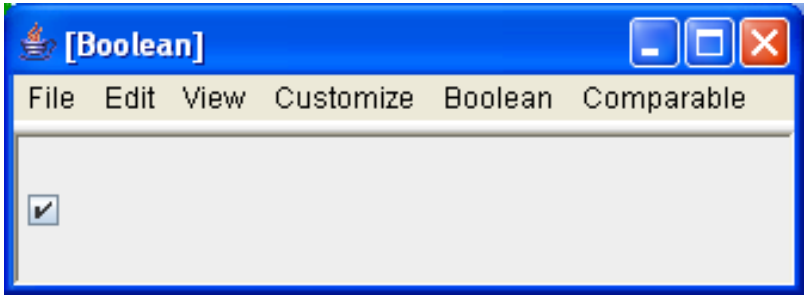


The Importance of Being Earnest

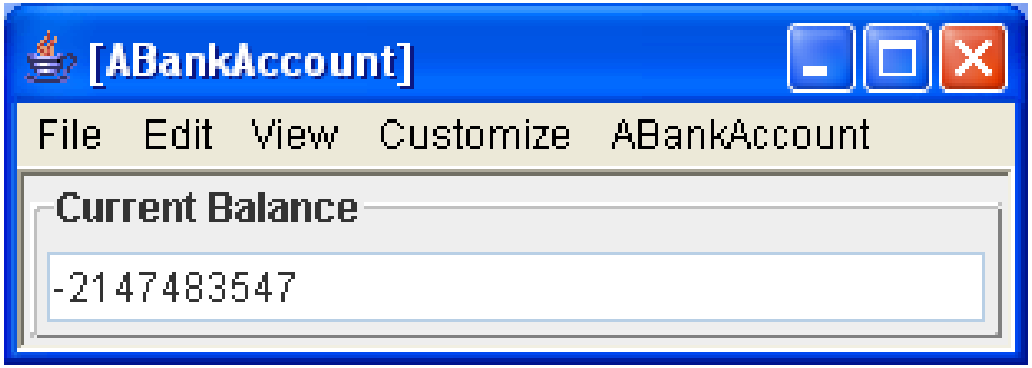




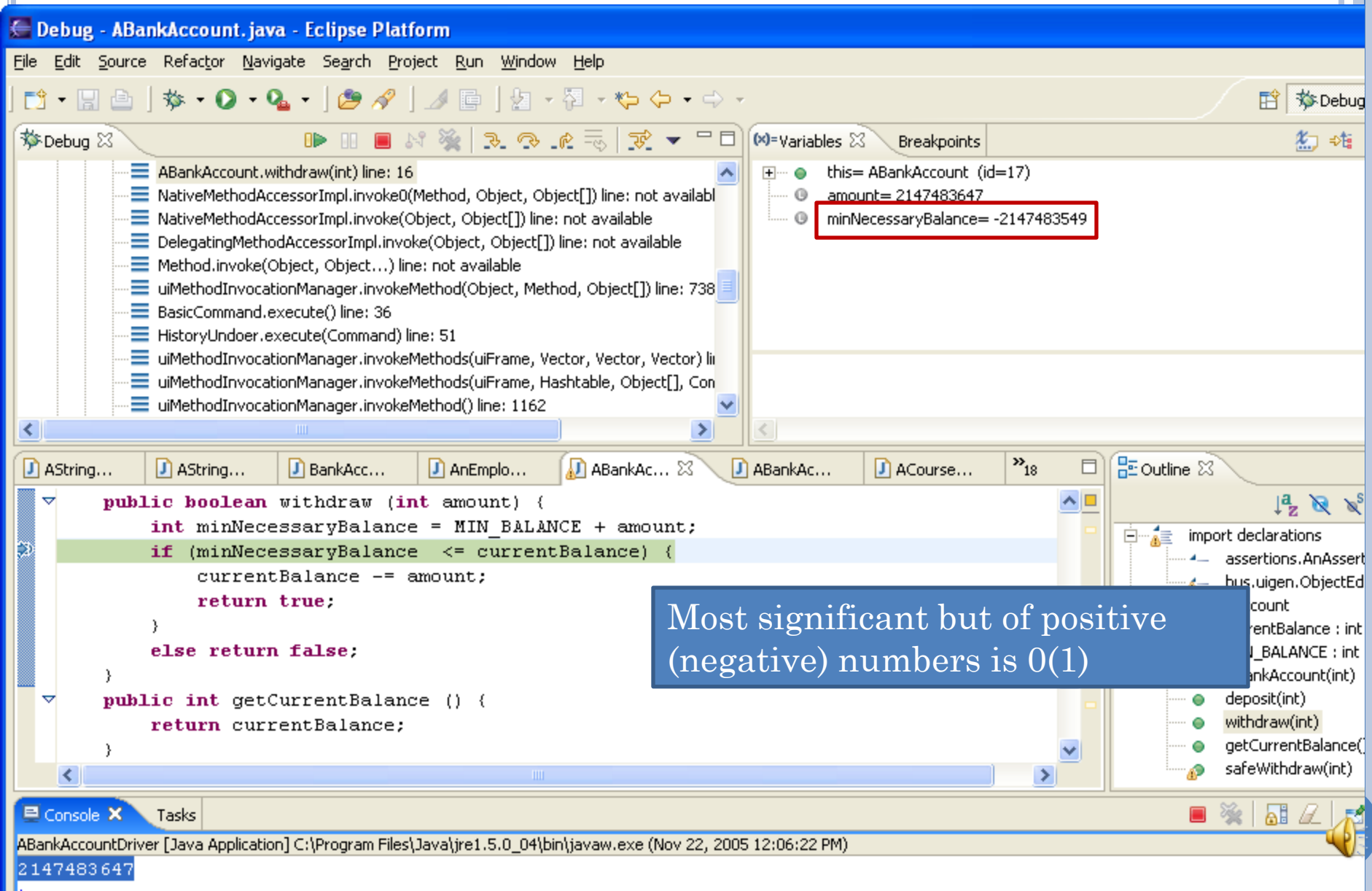
Integer.MAX_INT



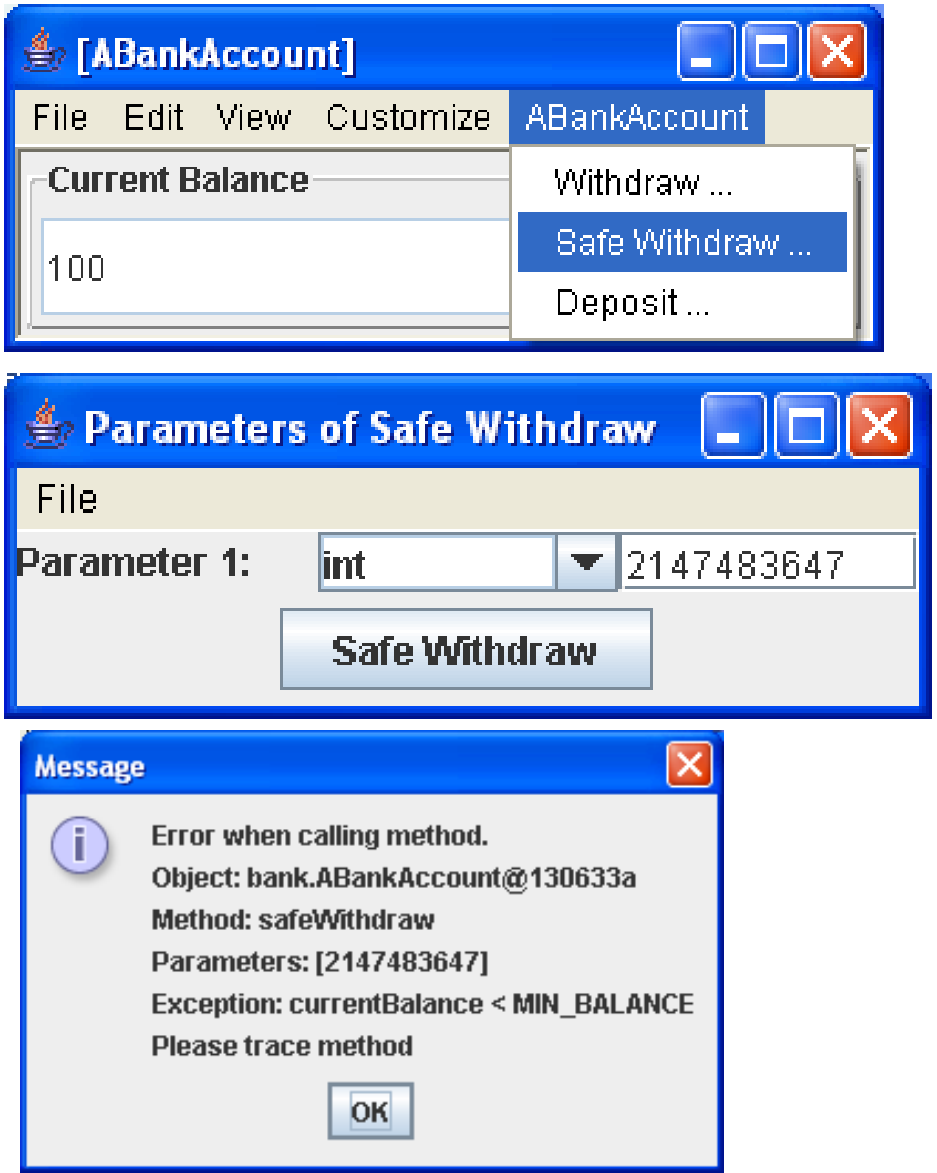
The Importance of Being Earnest



The Importance of Being Earnest



The Importance of Being Earnest



The Importance of Being Earnest

SecurityTracker.com Archives - Microsoft Internet Explorer **Integer** ...
Microsoft Internet Explorer **Integer Overflow** in Processing Bitmap Files Lets ...
Vendor URL: www.microsoft.com/technet/**security**/ (Links to External Site) ...
www.**security**tracker.com/alerts/2004/Feb/1009067.html - 25k - Cached - Similar pages

Security Issues - Gaim
Description, **Integer overflow** in memory allocation results in heap **overflow**.
By passing the size variable as ~0, **integer overflows** to 0 when 1 is added in ...
gaim.sourceforge.net/**security**/?id=2 - 7k - Cached - Similar pages

Gentoo Linux Documentation -- Samba: **Integer overflow**
Samba: **Integer overflow**. Content:. 1. Gentoo Linux **Security** Advisory, 2.
Impact Information, 3. Resolution Information, 4. References ...
www.gentoo.org/**security**/en/glsa/glsa-200412-13.xml - 9k - Cached - Similar pages

The KOffice Project - XPDF **Integer Overflow** 2
KOffice 1.3 (including betas) to 1.3.5 have an **integer overflow** vulnerability in
KWord's PDF ... References. the corresponding **security** advisiory for KDE. ...
www.koffice.org/**security**/2004_xpdf_**integer_overflow**_2.php - 9k - Cached - Similar pages

Microsoft Windows LoadImage API Function **Integer Overflow** ...
Microsoft Windows is reported prone to a remote **integer overflow** vulnerability.
... Microsoft Upgrade **Security** Update for Windows NT Server 4.0 (KB891711) ...
securityresponse.symantec.com/avcenter/**security**/Content/12095.html - 28k - Cached - Similar pages

CERT Advisory CA-2002-25 **Integer Overflow** In XDR Library
There is an **integer overflow** present in the xdr_array() function ... CERT publications
and other **security** information are available from our web site ...
www.cert.org/advisories/CA-2002-25.html - 26k - Cached - Similar pages

[LSS | **Security** | eXposed by LSS | Detalji]
LSS **Security** Advisory #LSS-2005-01-02. http://**security**.lss.hr. Title:, Apache
mod_auth_radius remote **integer overflow**. Advisory ID:, LSS-2005-01-02 ...
security.lss.hr/en/index.php?page=details&ID=LSS-2005-01-02 - 16k - Cached - Similar pages

Network **Security**, Vulnerability Assessment, Intrusion Prevention
eEye - Network **security** & vulnerability management software including ...
Although the copy length is similarly subject to an **integer overflow**, ...
www.eeye.com/html/research/advisories/AD20051108a.html - 18k - Nov 20, 2005 - Cached - Similar pages

Network **Security**, Vulnerability Assessment, Intrusion Prevention
For the purpose of signature development and further **security** research, a sample

Google for "integer overflow security"

Extra Slides

Problem with inaccessible variables

- Syntax
  - Qx:D(x):P(x)
  - ∀j: 0 <= j < b.size(): b.get(j) != null
  - ∃j: 0 <= j < b.size(): b.get(j) != null
- How to describe D(x) and P(x)?
  - Cannot pass expression string as variables involved have no meaning to library
  - Will pass one argument describing the domain
    - Collection of elements
  - Another argument describing the subproposition to be evaluated for each domain element.

Complete Asserter

```
package util.assertions;
import java.util.Iterator;
public class AnAsserter<ElementType> implements
Asserter<ElementType> {
    public void assertUniversal (Iterator<ElementType>
elements, ElementChecker elementChecker, String message) {
        while (elements.hasNext())
            if (!elementChecker.check(elements.next())) throw
new AssertionError (message);
    }
    public void assertExistential (Iterator<ElementType>
elements, ElementChecker elementChecker, String message) {
        while (elements.hasNext())
            if (elementChecker.check(elements.next()))
return;
        throw new AssertionError (message);
    }
}
```

Asserter Class

```
package util.assertions;
import java.util.Iterator;
public class AnAsserter<ElementType> implements Asserter<ElementType> {
    public void assertUniversal (Iterator<ElementType> elements, …, String
message) {
        assert checkUniversal(elements, …, message);
    }
    public void assertExistential (Iterator<ElementType> elements,
ElementChecker elementChecker, String message) {
        assert !checkExistential(elements, …, message)
    }
    public boolean checkUniversal (Iterator<ElementType> elements, …) {
        while (elements.hasNext())
            if (…) return false;
          return true;
    }
    public void checkExistential (Iterator<ElementType> elements, …) {
        while (elements.hasNext())
            if (…) return true;
          return false;
    }
}
```

Goal

```
package util.assertions;
import java.util.Iterator;
public interface Asserter<ElementType> {
    public void assertUniversal (Iterator<ElementType>
enumParam, ElementChecker elementChecker, String message);
    public void assertExistential (Iterator<ElementType>
enumParam, ElementChecker elementChecker, String message);
}
```

- Need to fill the …

Goal

```
package util.assertions;
public interface Asserter<ElementType> {
    public void assertUniversal (Iterator<ElementType>
enumParam, ElementChecker elementChecker, String message);
    public void assertExistential (Iterator<ElementType>
enumParam, ElementChecker elementChecker, String message);
}
```

Problem with inaccessible variables

- Syntax
  - Qx:D(x):P(x)
  - ∀j: 0 <= j < b.size(): b.get(j) != null
  - ∃j: 0 <= j < b.size(): b.get(j) != null
- How to describe D(x) and P(x)?
  - Cannot pass expression string as variables involved have no meaning to library
  - Will pass one argument describing the domain
    - Collection of elements
  - Another argument describing the subproposition to be evaluated for each domain element.

How to describe domain?

- Syntax
  - Qx:D(x):P(x)
  - ∀j: 0 <= j < b.size(): b.get(j) != null
  - ∃j: 0 <= j < b.size(): b.get(j) != null
- Domain can be
  - Array, Vector, StringHistory, …
- Need a common interface to describe elements
  - java.util.Iterator

Describing the Domain

```
public interface Asserter<ElementType> {
    public void assertUniversal (Iterator<ElementType>
domain, …, String message);
    public void assertExistential (Iterator<ElementType>
domain, …, String message);
}
```

```
asserter.assertUniversal (B.iterate(), …, "Some element of B is null");
asserter.assertExistential(B.iterate(), …, "All elements of B are null");
```

Need to fill …

Describing the Domain Using Method Parameters

```
public interface Asserter<ElementType> {
    public void assertUniversal (Iterator<ElementType>
domain, (<ElementType→ boolean)elementChecker , String
message);
    public void assertExistential (Iterator<ElementType>
domain, (<ElementType→ boolean)elementChecker, String
message);
}
```

asserter.assertUniversal (B.iterate(), …, "Some element of B is null");
asserter.assertExistential(B.iterate(), …, "All elements of B are null");

Need to fill …

Describing the Domain

```
public interface Asserter<ElementType> {
    public void assertUniversal (Iterator<ElementType>
enumParam, ElementChecker elementChecker, String message);
    public void assertExistential (Iterator<ElementType>
enumParam, ElementChecker elementChecker, String message);
}
```

Describing the domain

```
package assertions;
import java.util.Enumeration;
public class AQuantifier {
    public static boolean forAll ((Enumeration domain, …) {
            while (domain.hasMoreElements())
                    …
    }
    public static boolean thereExists ((Enumeration domain, …) {
            while (domain.hasMoreElements())
                    …
    }
}
```

AnAsserter.assert(AQuantifier.forAll(B.elements(), …, "Some element of B is null");

AnAsserter.assert(AQuantifier.thereExists(b.elements(), …, "All elements of B are null");

Need to fill …

How to describe Subproposition

- Syntax
  - Qx:D(x):P(x)
  - ∀j: 0 <= j < b.size(): b.get(j) != null
  - ∃j: 0 <= j < b.size(): b.get(j) != null
- Cannot pass expression string as variables involved have no meaning to library
- But can pass function that evaluates it.
  - ```
    boolean isNotNull(Object element) {
     return element != null;
    }
    ```
- Function will be evaluated for each domain element by our libraries
    ```

Subproposition as a function

```
package assertions;
import java.util.Enumeration;
public class AQuantifier {
    public static boolean forAll (Enumeration domain, (object → boolean)  subProposition) {
            while (domain.hasMoreElements())
                if (!subProposition (domain.nextElement()) return false;
            return true;
    }
    public static boolean thereExists ((Enumeration domain, (object → boolean)  subProposition) {
             while (domain.hasMoreElements())
                  if (!subProposition (domain.nextElement()) return true;
            return false;
    }
}
```

```
AnAsserter.assert(AQuantifier.forAll(B.elements(), isNotNull), "Some element of B is null");
AnAsserter.assert(AQuantifier.thereExists(b.elements(), isNotNull), "All elements of B are null");
```

How to describe Subproposition

```
import util.assertions;
public interface ElementChecker<ElementType> {
   public boolean check (ElementType element);
}
```

```
public class ANonNullChecker implements
ElementChecker<Object> {
  public boolean check(String element) {
       return element != null;
   }
}
```

t pass expression

n pass function
valuates it.

lean
tNull(Object
ent) {
return element != null;
}

does not support
tion parameters

allows object
meters
bject = data +
nctions

ubproposition
ts implement same
interface

- A subproposition object visits each element

Describing the subproposition

```
package assertions;
import java.util.Enumeration;
public class AQuantifier {
   public static boolean forAll (Enumeration domain, ElementChecker
subProposition) {
      while (domain.hasMoreElements())
         if (!subProposition.visit(domain.nextElement())) return false;
      return true;
   }
   public static boolean thereExists (Enumeration domain,
ElementChecker subProposition) {
      while (domain.hasMoreElements())
         if (subProposition.visit(domain.nextElement())) return true;
      return false;
   }
}
```

Call

Callback

```
AnAsserter.assert(AQuantifier.forAll(b.elements(), new
   ANonNullChecker()), "Some element of B is null");
AnAsserter.assert(AQuantifier.thereExists(b.elements(), new
   ANonNullChecker()), "All elements of B are null");
```

Goal

```
public interface Asserter<ElementType> {
    public void assertUniversal (Iterator<ElementType>
enumParam, ElementChecker elementChecker, String message);
    public void assertExistential (Iterator<ElementType>
enumParam, ElementChecker elementChecker, String message);
}
```

- Need to fill the …

Calls vs. Callbacks

- Calls
  - calls from reusing class to reused class
- Callbacks
  - calls from reused class to reusing class.
  - not to implement a symbiotic relationship
  - done to service calls

Subproposition accessing vars other than domain elements

- ∀j: 0 <= j < b.size(): b.get(j) != a.get(0)
- ∃j: 0 <= j < b.size(): b.get(j) != a.get(0)

```
public class ANonNullChecker implements
ElementChecker<Object> {
  public boolean check(String element) {
       return element != null;
  }
}
```

No constructor

AnAsserter.assert(AQuantifier.forAll(b.elements(), **new** ANonNullChecker()), "Some element of B is null");

Subproposition accessing vars other than domain elements

- ∀j: 0 <= j < b.size(): b.get(j) != a.get(0)
- ∃j: 0 <= j < b.size(): b.get(j) != a.get(0)

Each external var becomes constructor parameter

```
package util.assertions;
import assertions.ElementChecker;
public class AnInequalityChecker implements
ElementChecker<String> {
        Strring testObject;
        public AnInequalityChecker(String theTestObject) {
                testObject = theTestObject;
        }
        public boolean check(String element) {
                return !element.equals(testObject);
        }
}
```

```
AnAsserter.assert(AQuantifier.forAll(b.elements(), new
    AnInequalityChecker(a.get(0))), "Some element of b is equal to
    a.get(0)");
```

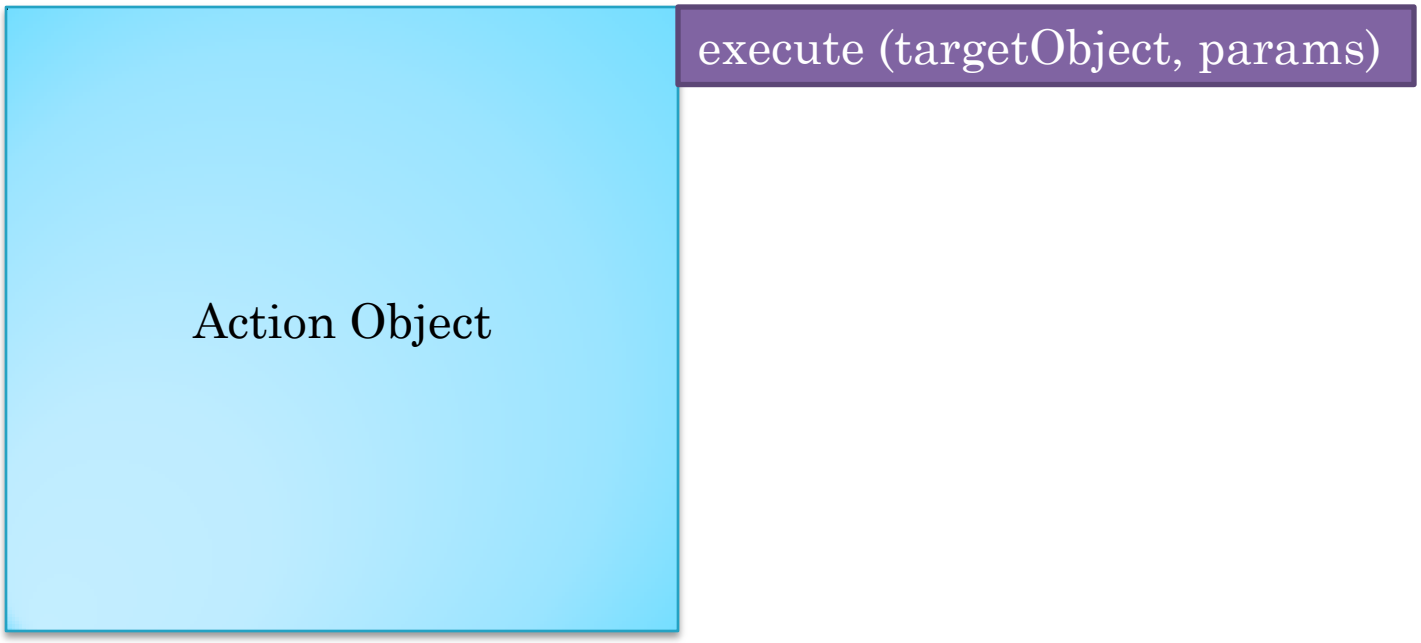
Visitor Pattern

- Some collection C of elements of type T
- Visitor interface
  - **public interface** ElementChecker { **public boolean** visit (Object element);}
  - **public interface** V {**public** T2 m (T p);}
- One or more traverser methods that use collection and visitor interface to pass one or more collection elements to the method.
  - **public static boolean** forAll (Enumeration domain, ElementChecker subProposition) {…subProposition.visit(domain.nextElement());
  - traverser1 (C c, V v) { …v.m(element of C)…}
- Implementation of interface whose constructors take as arguments external variables that need to be accessed by the visitor method

```
public class AnInequalityChecker implements ElementChecker {
    Object testObject;
    public AnInequalityChecker(Object theTestObject) {…}
    public boolean visit(Object element) {…};
}
public class AV1 implements V {
    public AV1 (T1 p1, … Tn pN) { …}
    public T2 m (T p) { … }
}
```

- Client passes traverser visitor implementation and collection
  - AQuantifier.forAll(b.elements(), **new** AnInequalityChecker(a.get(0)));
  - traverser1(c, **new** AV1(a1,.. aN));

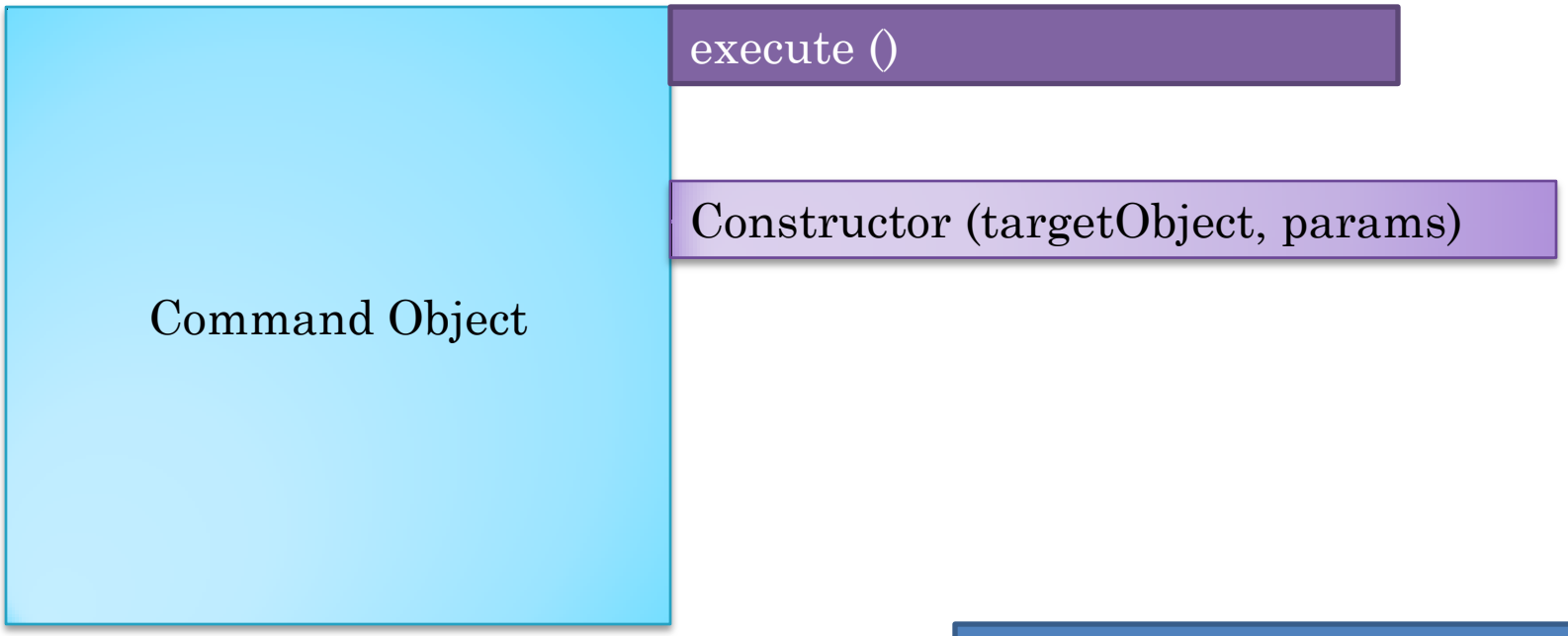
ActionObject



Provides an execute operation to perform some action.

The execute operation takes the object on which the target operation is to invoked and an array of parameters of the target method.

Command Object



Provides a execute operation to perform some action.

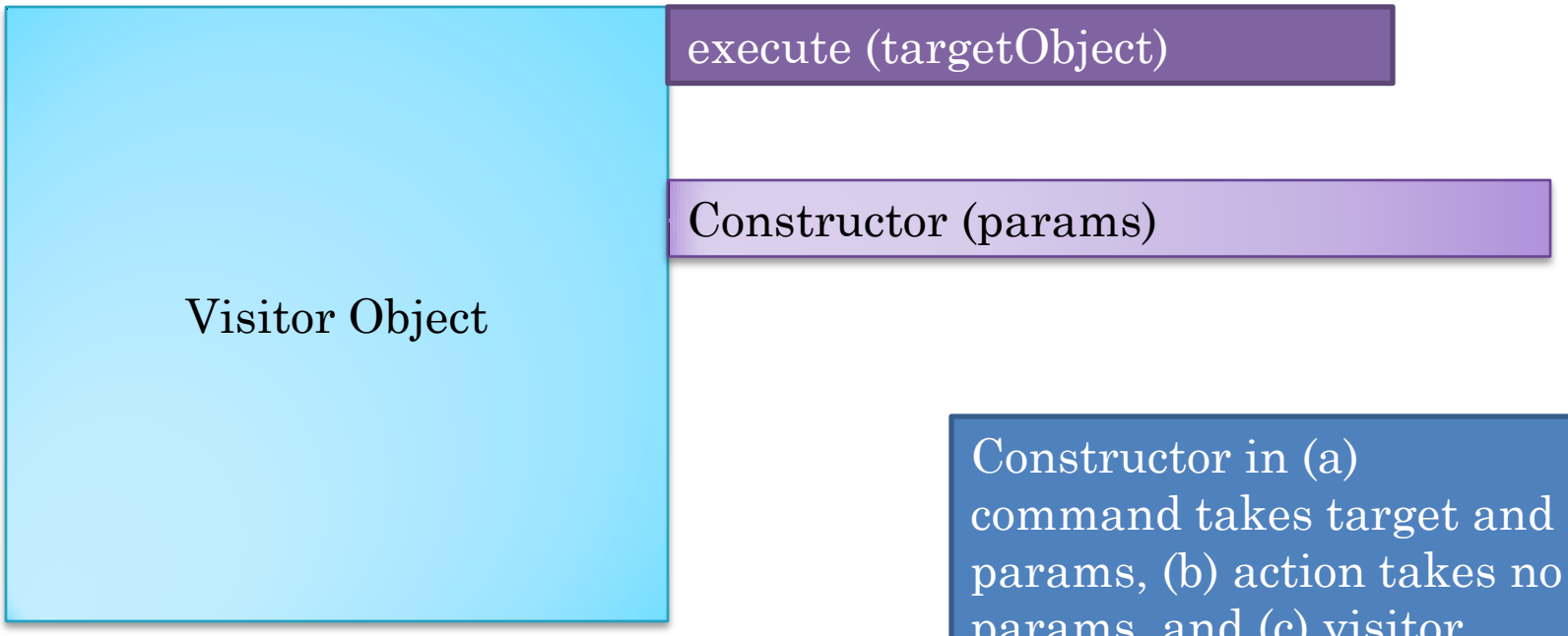
The execute operation takes no arguments.

Constructor takes parameters of operation as arguments.

Action is an operation that can be invoked on many different arguments

A command is a specific action invocation.

Visitor Object



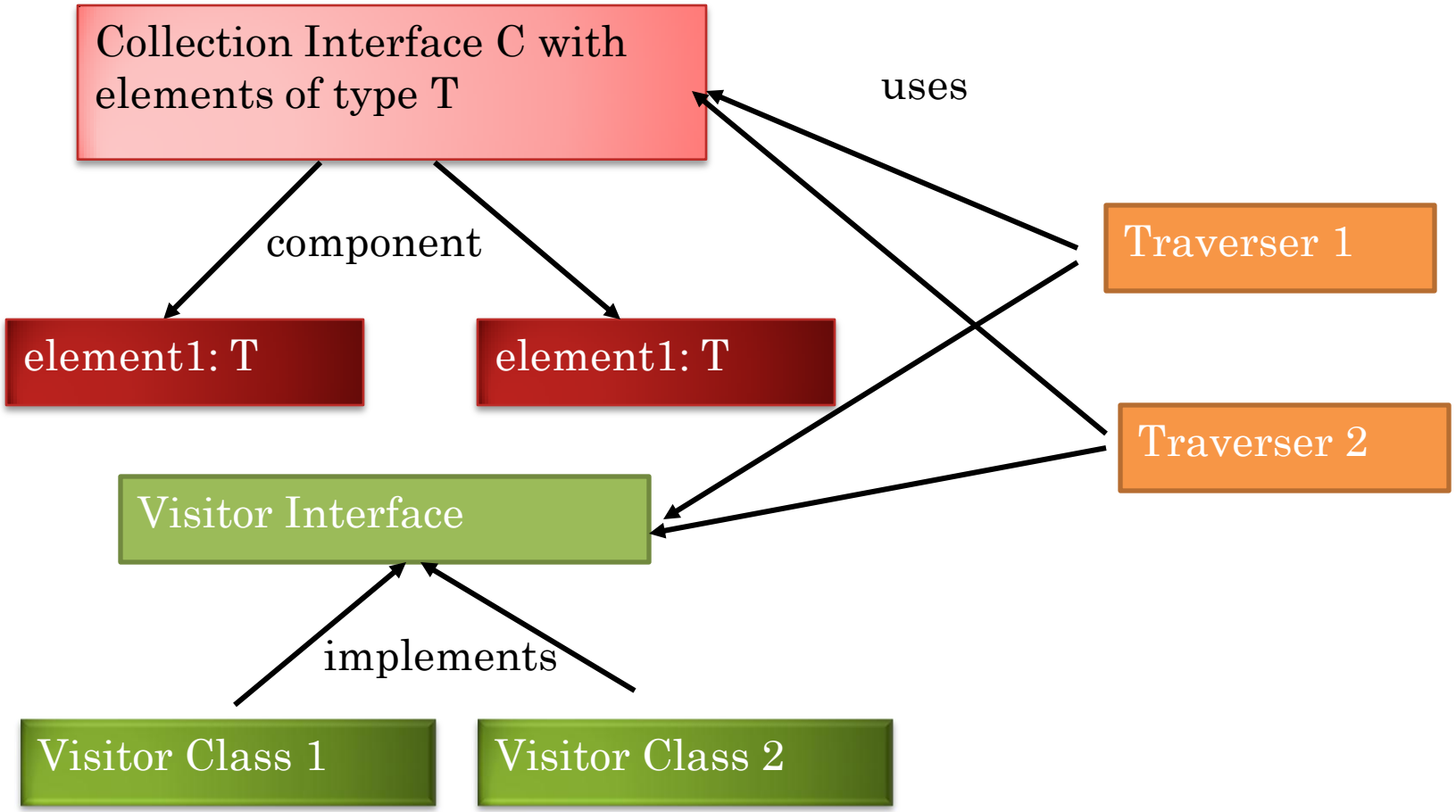
Provides a execute operation to perform some action.

The execute operation takes target object as argument

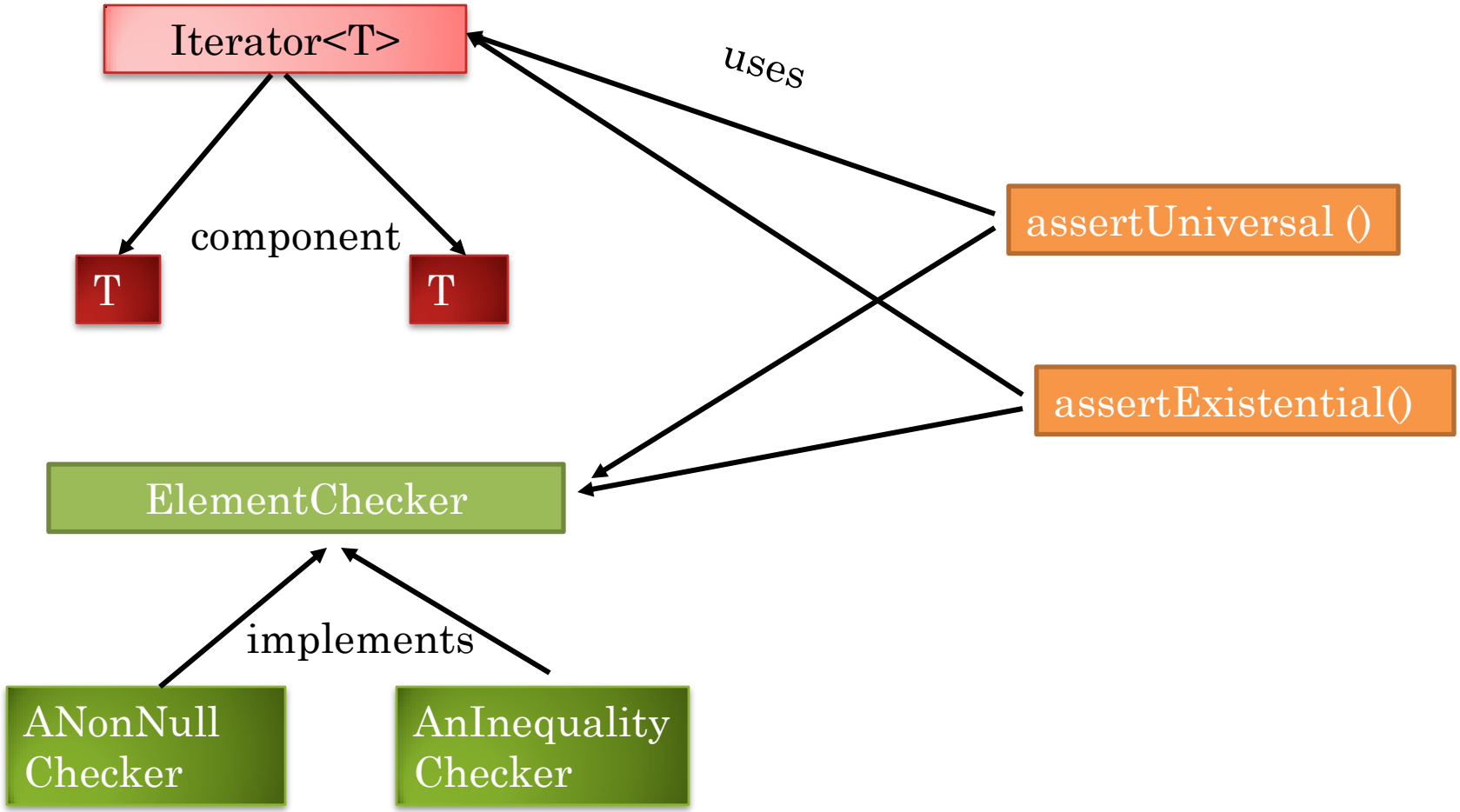
Constructor in (a) command takes target and params, (b) action takes no params, and (c) visitor takes oparams as arguments.

Execute in (a) command takes no params, (b) action takes target object and params, (c) visitor takes target object

Visitor Pattern



Example of Visitor Pattern



Nested Assertions

∀j: 0 <= j < b.size(): ∀k: 0 <= k < b.get(j).size(): b.get(j).get(k) != null

```
package visitors;
import assertions.ElementChecker;
public class ANonNullChecker implements ElementChecker {
    public boolean visit(Object element) {
            return (element != null);
    }
}
```

Visiting Hierarchical Structures (Trees)

```
package visitors;
import assertions.ElementChecker;
public class AListChecker implements ElementChecker<List> {
    public boolean check(List element) {
        Iterator children = element.iterate();
        return asserter.assertUniversal(children, new
ANonNullChecker()));
    }
}
```

```
AnAsserter.assert(AQuantifier.forAll(b.elements(), new AForAllChecker(), "Some leaf-level
    element of b is null");
```

Example of Pattern in Everyday Applications

- Program tree
- A visitor for printing all nodes.
- Another for type checking.
- Yet another for generating code
- Do not want to put all of this code in tree class.
- In any case, printing should not be in tree.

Subproposition accessing vars other than domain elements

```
public class ANonNullChecker implements
ElementChecker<Object> {
  public boolean check(Object element) {
       return element != null;
  }
}
```

No Parameterized constructor

∀j: 0 <= j < b.size(): b.get(j) != a.get(0)

assert checkUniversal(b.iterator(), **new** ANonNullChecker()): "some element of b is null")

Check Method

```
public boolean nonNullChecker(Object element) {
        return element != null;
}
```