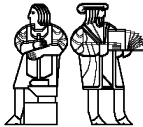


6.170 Lecture 8 Specifications and Exceptions



John Guttag
MIT EECS

6.170 Specs and Implementations, A Quick Review

Consider two specifications, S1 and S2
Assume that S1 is stronger than S2
Every implementation of S1
Is also an implementation of S2
If a program that calls S2 works
It would also work if it called S1
This is the axiom from which everything follows

© Michael Ernst/John Guttag Spring 2001 Slide 2

6.170 What Should It Mean When Something Omitted?

requires \Rightarrow (modifies \wedge effects)

What would implementer prefer?
Remove constraints on implementation
requires false
modifies anything
effects true

What would user of procedure prefer?
Useful in the broadest variety of contexts
requires true
No unanticipated side-effects
modifies nothing
No obvious answer for effects

© Michael Ernst/John Guttag Spring 2001 Slide 3

6.170 Default Values

Requires
Purists view, expressed in book
requires true
Pragmatists view, what we expect you to do
default: requires “no null pointers”

modifies
modifies nothing

effects
Should not be omitted

© Michael Ernst/John Guttag Spring 2001 Slide 4

6.170 Non-empty Requires

```
public static int zero(int x)
  requires x > 0
  effects returns 0
-----
public static int zero(int x) {
  while (x != 0) x = x - 1;
  return x; }

public static int zero(int x) {
  if (x == 0) return 1;
  return 0; }
```

Does each program implement the specification ?

© Michael Ernst/John Guttag Spring 2001 Slide 5

6.170 Impact on callers Callers

```
public static int zero(int x)
  requires x > 0
  effects returns 0
```

Consider a program that calls this procedure

```
public static boolean caller (int x, int y) {
  return (zero(x) == zero(y)); }
```

What does caller do ?

© Michael Ernst/John Guttag Spring 2001 Slide 6

The Moral

Partial procedures make the world more complicated

GEFAHR
Peligro!
DANGER
危険

© Michael Ernst/John Guttag Spring 2001 Slide 7

What Should One Do?

Ideally, avoid partial procedures

Easy in this case, merely remove requires
Yields a stronger specification
Abstraction useful in more circumstances

Is there any reason not to do that here?
Does restrict implementations
But I can't think of a good reason to care

© Michael Ernst/John Guttag Spring 2001 Slide 8

A More Interesting Example

Return the i^{th} element of a list
Suppose the list has fewer than i elements ?

© Michael Ernst/John Guttag Spring 2001 Slide 9

One Possibility

```
Integer IntegerList.fetch(int i)
  requires i>0 & this has at least i
         elements
  effects returns the ith element of this
-----
if (i > 0 && s.length() > i)
  j = s.fetch(i);
```

Why is this bad?
Error prone, suppose caller forgets to test length?
Anything goes -- crash, loop, return 6 ...
Inefficient
Two procedure calls

© Michael Ernst/John Guttag Spring 2001 Slide 10

Better to Make Procedure Total

Total means well-defined everywhere

Two ways to do this
Return value that is in type but special
Null often used for this in Java
-1 frequently used in C
Use exceptions

© Michael Ernst/John Guttag Spring 2001 Slide 11

Using a Special Value

```
Integer IntegerList.fetch(int i)
  effects
  i>0 & size(this)>i:
    returns ith element of this
  otherwise: returns null
```

Why is this not great?
Sometimes no extra value
Caller must check if special value has been returned
Ugly
Inefficient
Error prone

© Michael Ernst/John Guttag Spring 2001 Slide 12

Using an Exception

```
Integer IntegerList.fetch(int i)
  throws NoSuchElementException
  effects
  i > 0 & size(this) > i:
    returns ith element of this
  otherwise: throws NoSuchElementException
-----
try {
  j = s.fetch(i);
} catch (NoSuchElementException e) {
  Whatever one wants to do
}
//continue for both cases
```

© Michael Ernst/John Guttag Spring 2001 Slide 13

Catching Exceptions

Caught by catch associated with nearest enclosing try

If no such catch
Exception propagated up call stack

If not caught at all
Program terminates

© Michael Ernst/John Guttag Spring 2001 Slide 14

Two Kinds of Exceptions

Checked exceptions
E.g., class Missing extends Exception
Compiler error unless
There exists a catch clause
Caller declared to throw that exception

Unchecked exceptions
E.g., class ArithmeticException extends RuntimeException
Compiler doesn't complain

Rule of thumb
Stick to checked exceptions most of the time
Use unchecked exceptions to mean failure
Expect program to terminate

© Michael Ernst/John Guttag Spring 2001 Slide 15

Why Catch Exceptions Locally?

Failure to catch exceptions violates modularity
A → IntegerSet.insert → IntegerList.insert
IntegerList.insert throws an exception
Implementor of IntegerSet.insert knows how list is being used
Implementor of A may not even know that IntegerList exists

Procedure up the line may think that it is handling an exception raised by a different call

Even if exception is better handled up a level
Probably better to catch it and throw it again
Makes it clear to reader of code that it was not an omission

© Michael Ernst/John Guttag Spring 2001 Slide 16

Exceptions in Review

Use an exception when
Used in a broad or unpredictable context
Checking the condition is feasible

Use a precondition when
Checking would be prohibitive
E.g., requiring that a list be sorted
Used in a narrow context in which calls can be checked

Preconditions should be avoided because
Caller may violate precondition
Program can fail in uninformative or dangerous way
Want program to fail as early as possible

© Michael Ernst/John Guttag Spring 2001 Slide 17

Exceptions in Review, cont.

Use checked exceptions most of the time

Handle exceptions sooner rather than later

Don't think of them as errors
A program structuring mechanism

Documentation standards and conventions
Will be covered in recitation

© Michael Ernst/John Guttag Spring 2001 Slide 18