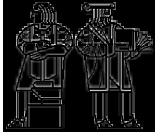


6.170 Lecture 6 Specifications and Exceptions



John Guttag
MIT EECS



If 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



Which Is Stronger?

```
public static int test(Vector v, Object o, Object o1)
  requires o occurs in v
  modifies v
  effects returns an i such that ith element of v is o
    & sets the ith element of v to o1
    & makes no other changes to v
```

```
public static int test(Vector v, Object o, Object o1)
  requires o occurs in v
  modifies v
  effects returns an i such that ith element of v is o
    & sets the ith element of v to o1
```



Which Is Stronger?

```
public static int test(Vector v, Object o, Object o1)
  requires true
  modifies v
  effects returns an i such that ith element of v is o
    & sets the ith element of v to o1
    & makes no other changes to v
```

```
public static int test(Vector v, Object o, Object o1)
  requires o occurs in v
  modifies v
  effects returns an i such that ith element of v is o
    & sets the ith element of v to o1
    & makes no other changes to v
```



What Should It Mean When Something Omitted?

requires ⇒ (**modifies** & **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



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

6.170 Spring 2000

A Last Example

<p>S</p> <pre>int id (int x); effects returns x</pre>	<p>P</p> <pre>int id (int x) { while (true);}</pre>
--	--

Recall that we said that P satisfies S iff
 $P \mid (\text{Domain of } S) \text{ is a subset of relation denoted by } S$

Does P satisfy S?
 $S = \{ \langle 0, 0 \rangle, \langle 1, 1 \rangle, \langle -1, -1 \rangle, \dots \}$
 $P = \{ \}$

$P \mid \{x: x \text{ is an int}\} \not\subseteq S$
 Seems to suggest yes

But what does your ♥ tell you?

© John Chapin/John Guttag Spring 2000 Slide 7

P Satisfies S, the Truth

P satisfies S iff
 $P \mid (\text{Domain of } S) \text{ is a subset of relation denoted by } S$
 $\& x \in \text{Domain of } S \Rightarrow \exists y \text{ s.t. } \langle x, y \rangle \in P$

I.e., P must be defined on every value in the domain of S

What is the moral here?
 Partial procedures make the world more complicated

**GGFARE
Poligral
DANGER
危険**

© John Chapin/John Guttag Spring 2000 Slide 8

Unfortunately, Partial Procedures Seem Natural

```
Integer IntegerList.fetch(int i)
  requires this has at least i elements
  effects returns the ith element of this
```

```
if (s.length() > i) j = s.fetch(i);
```

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

© John Chapin/John Guttag Spring 2000 Slide 9

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

© John Chapin/John Guttag Spring 2000 Slide 10

Using a Special Value

```
Integer IntegerList.fetch(int i)
  effects
  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


© John Chapin/John Guttag Spring 2000 Slide 11

Using an Exception

```
Integer IntegerList.fetch(int i)
  throws NoSuchElementException
  effects
  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
```

© John Chapin/John Guttag Spring 2000 Slide 12


 **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

© John Chapin/John Guttag Spring 2000 Slide 13


 **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

© John Chapin/John Guttag Spring 2000 Slide 14


 **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

© John Chapin/John Guttag Spring 2000 Slide 15


 **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

© John Chapin/John Guttag Spring 2000 Slide 16

 **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

© John Chapin/John Guttag Spring 2000 Slide 17