CSC 520, Spring 2020

Principles of Programming Languages

Michelle Strout

THE UNIVERSITY
OF ARIZONA

Version: 1

April 20, 2020

Plan

- **Announcements**
  - HW9 is due Wednesday April 22nd
  - HW10 was posted last Friday and is due Wednesday April 29th
- **Last time**
  - Some questions about deriving type constraints (see piazza post)
  - Intro to Object-Oriented Programming
- **Today**
  - Six questions about uSmalltalk
  - Message passing
  - Dynamic dispatch
  - Predefined number classes

Slide Content Credits: Tufts Comp105 by Norman Ramsey and Kathleen Fisher

Mechanisms review

- Message send replaces function application
- Receiver appears first: it's in charge
- Respond to message by evaluating method
- Method determined by an object's class
- A method can use primitive operations

Slide Content Credits: Tufts Comp105 by Norman Ramsey and Kathleen Fisher

Six questions answered for uSmalltalk

- 1. What are the values?
- 2. What is the concrete and abstract syntax?
- 3. What are the environments?
- 4. What are the types?
- 5. What are the operational/dynamic semantics?
- 6. What is in the initial basis?

Slide Content Credits: Tufts Comp105 by Norman Ramsey and Kathleen Fisher

Six questions about Smalltalk

- **1. Values are objects**
  - Even true, 3, and “hello” are objects
  - Even classes are objects
  - There are no function values, only methods on objects

- **2. Syntax**
  - Mutable variables
  - Message send
  - Sequential composition of mutations and message sends (side effects)
  - “Blocks” (really closures, objects and closures in one, used as continuations)
  - No `if` or `while`, These are implemented by passing continuations to Boolean objects.

Slide Content Credits: Tufts Comp105 by Norman Ramsey and Kathleen Fisher

Syntax comparison: Impcore

```
Exp = LITERAL of value
    | VAR     of name
    | SET     of name * exp
    | IF      of exp * exp * exp
    | WHILE   of exp * exp
    | BEGIN   of exp list
    | APPLY   of name * exp list
```

Slide Content Credits: Tufts Comp105 by Norman Ramsey and Kathleen Fisher

Syntax comparison: Smalltalk

```
Exp = LITERAL of rep
    | VAR     of name
    | SET     of name * exp
~~    | IF      of exp * exp * exp~~
~~    | WHILE   of exp * exp~~
    | BEGIN   of exp list
~~    | APPLY   of name * exp list~~
    | SEND    of exp * name * exp list
    | BLOCK   of name list * exp list
```

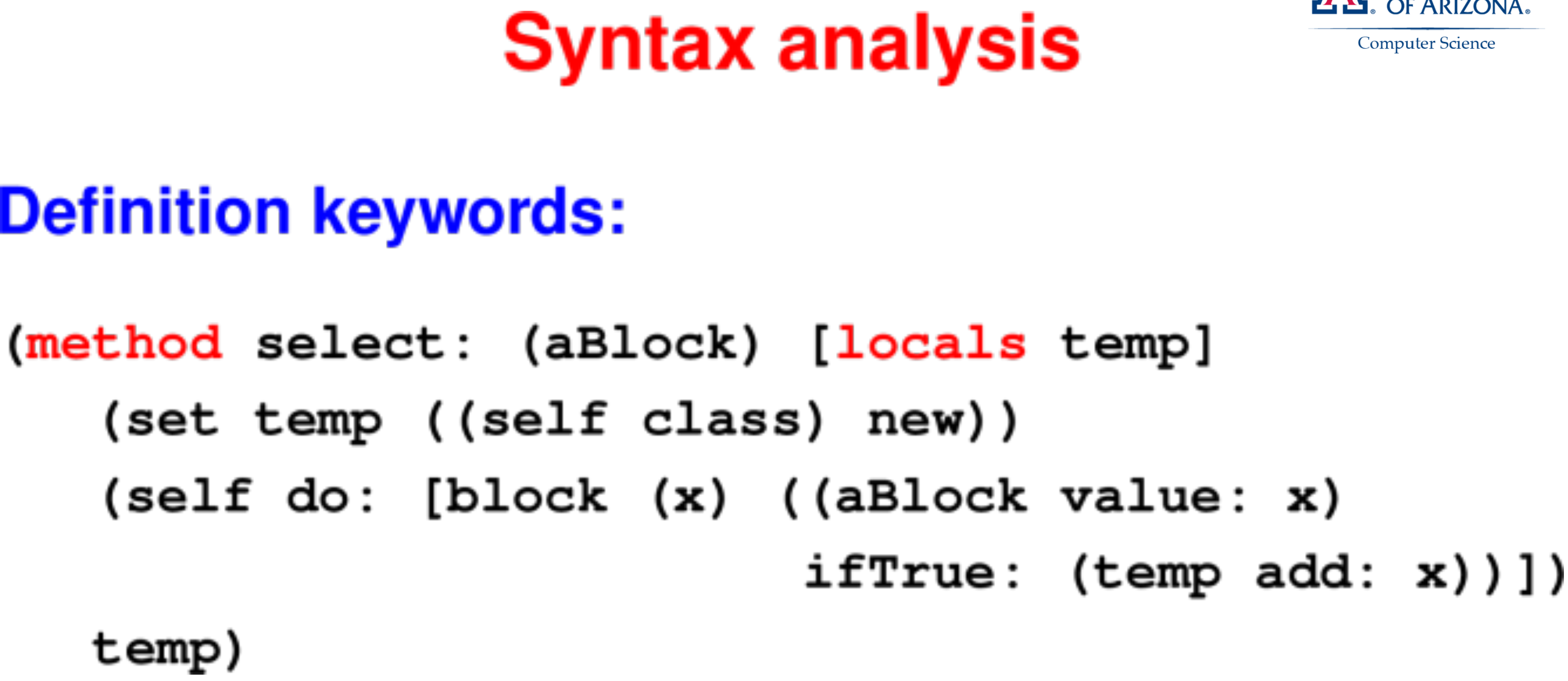
Syntax comparison: Smalltalk

```
Exp = LITERAL of rep
    | VAR     of name
    | SET     of name * exp
~~  | IF      of exp * exp * exp~~
~~  | WHILE   of exp * exp~~
    | BEGIN   of exp list
~~  | APPLY   of name * exp list~~
    | SEND    of exp * name * exp list
    | BLOCK   of name list * exp list
```

Slide Content Credits: Tufts Comp105 by Norman Ramsey and Kathleen Fisher

Message passing

- Look at SEND
  - Message identified by name (messaged are not values)
  - Always sent to a receiver
  - Optional arguments must match arity of message name (no other static checking)
- Note: BLOCK and LITERAL are special objects



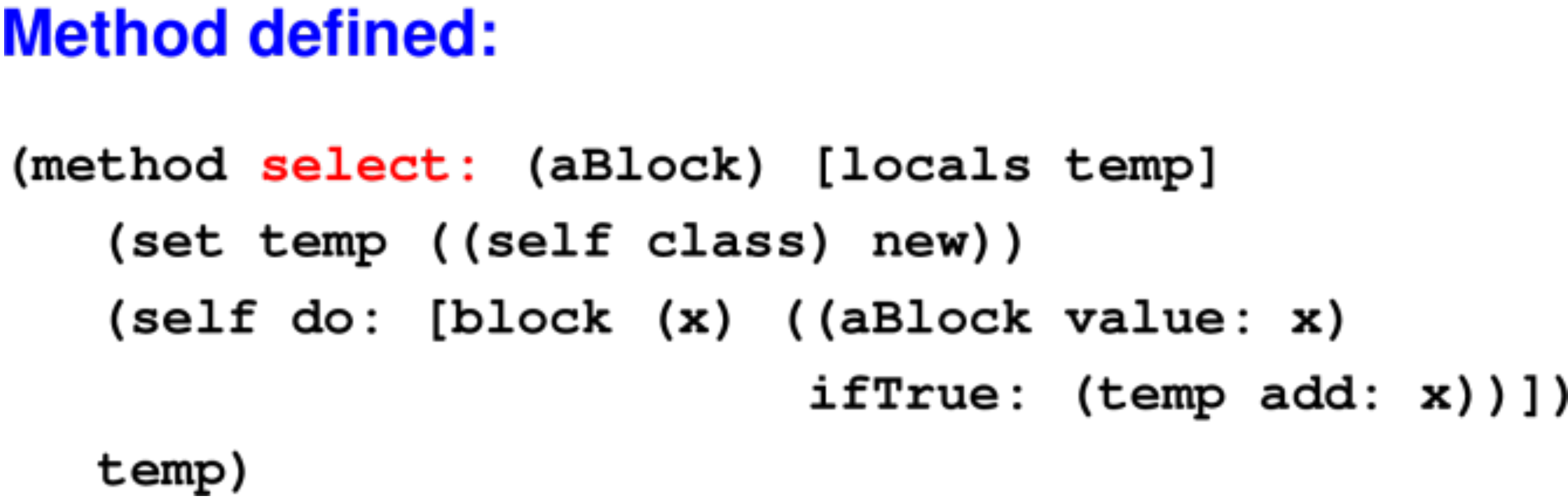
Syntax analysis

Definition keywords:

```
(method select: (aBlock) [locals temp]
   (set temp ((self class) new))
   (self do: [block (x) ((aBlock value: x)
                          ifTrue: (temp add: x))])
   temp)
```

Slide Content Credits: Tufts Comp105 by Norman Ramsey and Kathleen Fisher

Syntax analysis



Method defined:

```
(method select: (aBlock) [locals temp]
   (set temp ((self class) new))
   (self do: [block (x) ((aBlock value: x)
                         ifTrue: (temp add: x))])
  temp)
```

Slide Content Credits: Tufts Comp105 by Norman Ramsey and Kathleen Fisher

Syntax analysis

Expression form—messages sent:

```
(method select: (aBlock) [locals temp]
   (set temp ((self class) new))
   (self do: [block (x) ((aBlock value: x)
                         ifTrue: (temp add: x))])
   temp)
```

Slide Content Credits: Tufts Comp105 by Norman Ramsey and Kathleen Fisher

Syntax analysis

THE UNIVERSITY OF ARIZONA
Computer Science

Variables defined:

```
(method select: (aBlock) [locals temp]
   (set temp ((self class) new))
   (self do: [block (x) ((aBlock value: x)
                          ifTrue: (temp add: x))])
   temp)
```

Slide Content Credits: Tufts Comp105 by Norman Ramsey and Kathleen Fisher

Syntax analysis

Expression form—set:

```
(method select: (aBlock) [locals temp]
   (set temp ((self class) new))
   (self do: [block (x) ((aBlock value: x)
                         ifTrue: (temp add: x))])
  temp)
```

Slide Content Credits: Tufts Comp105 by Norman Ramsey and Kathleen Fisher

Syntax analysis

Expression form—`block`:

```
(method select: (aBlock) [locals temp]
   (set temp ((self class) new))
   (self do: [block (x) ((aBlock value: x)
                          ifTrue: (temp add: x))])
   temp)
```

Slide Content Credits: Tufts Comp105 by Norman Ramsey and Kathleen Fisher

Six questions about Smalltalk

• 3. Environments

- Name stands for a mutable cell containing an object:
  - Global variables
  - Formal parameters
  - Local variables
  - "Instance variables" (new idea, read about in Ramsey book)

• 4. Types

- There is no compile-time type system
- At runtime, Smalltalk uses behavioral subtyping, known also as "duck typing"
- Note that subclassing and subtyping are not equivalent. Think interfaces in Java.

Slide Content Credits: Tufts Comp105 by Norman Ramsey and Kathleen Fisher

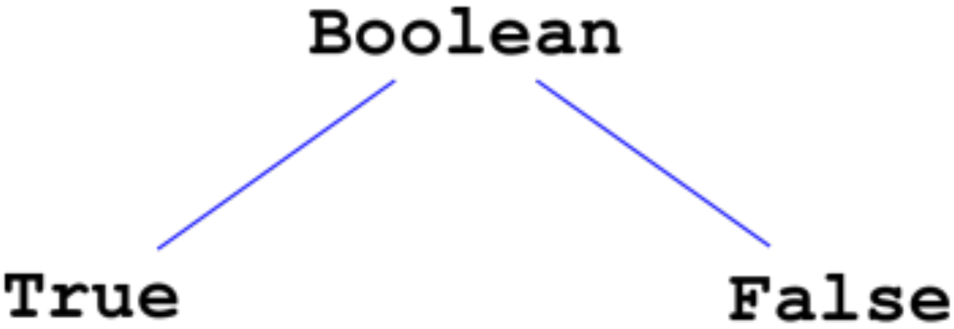
Six questions about Smalltalk

- 5. Dynamic semantics
  - Main rule is method dispatch (complicated). To answer a message:
    - Consider the class of the receiver
    - Is the method with that name defined?
    - If so, use it.
    - If not, repeat with the superclass.
    - Run out of superclasses? “Message not understood”
  - The rest is familiar

- 6. The initial basis is enormous
  - Why? To demonstrate the benefits of reuse, you need something big enough to reuse.
  - Also, everything is an object, including booleans and numbers.

Slide Content Credits: Tufts Comp105 by Norman Ramsey and Kathleen Fisher

Review: Inheritance for Booleans



`Boolean` is an abstract class

- Instances of `True` and `False` only

Method "`ifTrue:ifFalse:`" defined in `True` and `False`

All others defined in `Boolean`

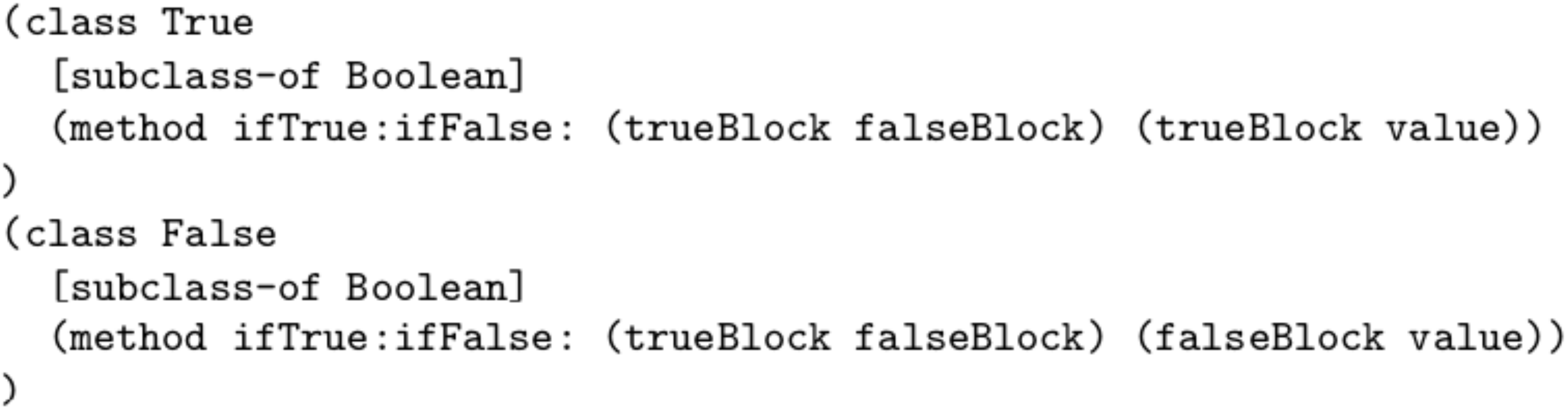
Slide Content Credits: Tufts Comp105 by Norman Ramsey and Kathleen Fisher

Protocol for Booleans

`ifTrue:ifFalse: trueBlock falseBlock`	Full conditional
`ifTrue: trueBlock`	Part conditional (for side effect)
`ifFalse: falseBlock`	Part conditional (for side effect)
`& aBoolean`	Conjunction
`| aBoolean`	Disjunction
`not`	Negation
`eqv: aBoolean`	Equality
`xor: aBoolean`	Difference
`and: altBlock`	Short-circuit conjunction
`or: altBlock`	Short-circuit disjunction

Slide Content Credits: Tufts Comp105 by Norman Ramsey and Kathleen Fisher

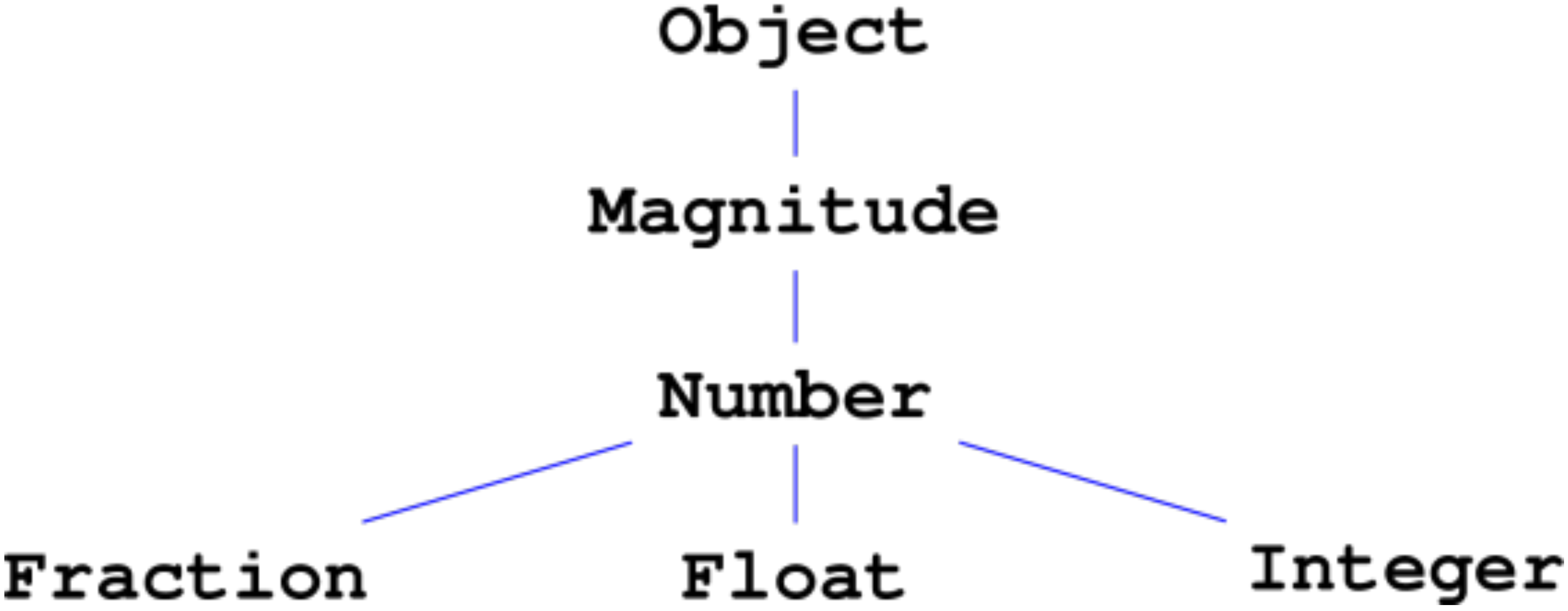
Implementation of ifTrue:ifFalse:



```
(class True
  [subclass-of Boolean]
  (method ifTrue:ifFalse: (trueBlock falseBlock) (trueBlock value))
)
(class False
  [subclass-of Boolean]
  (method ifTrue:ifFalse: (trueBlock falseBlock) (falseBlock value))
)
```

Slide Content Credits: Tufts Comp105 by Norman Ramsey and Kathleen Fisher

“Number hierarchy”



Object

Magnitude

Number

Fraction

Float

Integer

Slide Content Credits: Tufts Comp105 by Norman Ramsey and Kathleen Fisher

Each class has one of two roles

- **Abstract class**
  - Meant to be inherited from
  - Some (>0) subclassResponsibility methods
  - Examples: Boolean, Shape, Collection
- **Regular (“concrete”) class**
  - Meant to be instantiated
  - No subclassResponsibility methods
  - Examples: True, Triangle, List, SmallInteger

Slide Content Credits: Tufts Comp105 by Norman Ramsey and Kathleen Fisher

Instance protocol for `Magnitude`

`= aMagnitude`	equality (like Magnitudes)
`< aMagnitude`	comparison (ditto)
`> aMagnitude`	comparison (ditto)
`<= aMagnitude`	comparison (ditto)
`>= aMagnitude`	comparison (ditto)
`min: aMagnitude`	minimum (ditto)
`max: aMagnitude`	maximum (ditto)

Subclasses: `Date`, `Natural`

- Compare `Date` with `Date`, `Natural` w/`Natural`, `...`

Slide Content Credits: Tufts Comp105 by Norman Ramsey and Kathleen Fisher

Implementation of Reuse

- **Note: assume = and < implemented in subclass**

- **But if can’t get the form of the parameter how?**

```
(class Magnitude ; abstract class
    [subclass-of Object]
    (method = (x) (self subclassResponsibility))
                   ; may not inherit = from Object
    (method < (x) (self subclassResponsibility))
    (method > (y) (y < self))
    (method <= (x) ((self > x) not))
    (method >= (x) ((self < x) not))
    (method min: (aMag)
           ((self < aMag) ifTrue:ifFalse: {self} {aMag}))
    (method max: (aMag)
           ((self > aMag) ifTrue:ifFalse: {self} {aMag}))
)
```

Slide Content Credits: Tufts Comp105 by Norman Ramsey and Kathleen Fisher

One way: uSmalltalk primitives

```
(class SmallInteger
    [subclass-of Integer] ; primitive representation
    (class-method new: (n) (primitive newSmallInteger self n))
    (class-method new  ()  (self new: 0))
    (method negated    ()  (0 - self))
    (method print      ()  (primitive printSmallInteger self))
    (method +          (n) (primitive + self n))
    (method -          (n) (primitive - self n))
    (method *          (n) (primitive * self n))
    (method div:       (n) (primitive div self n))
    (method =          (n) (primitive sameObject self n))
    (method <          (n) (primitive < self n))
    (method >          (n) (primitive > self n))
)
```

Slide Content Credits: Tufts Comp105 by Norman Ramsey and Kathleen Fisher

Summary of Key Ideas

- **Protocol determines behavioral subtyping**
  - The protocol of an object is the set of messages it understands.
  - Object A is a behavioral subtype of Object B if A understand all of the messages that B does in a compatible way.
  - Intuition: If A is a behavorial subtype of B, then A can be used in any context where B can be used.
- **Class-based object-orientation**
  - Object implementations determined by its class definition
  - So, each class implicitly defines the protocol for its objects and dynamic dispatch is determined by object's class
  - Code reuse by sending messages around like crazy

Slide Content Credits: Tufts Comp105 by Norman Ramsey and Kathleen Fisher

Summary of Key Ideas cont…

- **What's hard**
  - Encapsulation: abstraction function and invariant
  - Higher-order programming: everything is higher order
  - Dynamic dispatch: every call is to an unknown function (trust the contract)
  - Inheritance: big vocabulary, hard to work on one function in isolation
  - Net effect: algorithms ”smeared out” over many methods
- **What's great**
  - Each method is super simple
  - Cooperating-objects model
  - Reuse, reuse, reuse

Slide Content Credits: Tufts Comp105 by Norman Ramsey and Kathleen Fisher