CSCI-344
Programming Language Concepts (Section 3)

Lecture 19

More Smalltalk (inheritance, block)

Instructor: Hossein Hojjat

October 21, 2016

Where we are

Done:

- Smalltalk: an object-oriented language

This session:

- More Smalltalk (inheritance, block)

Protocol

- Protocol: the interface to an object
- Object's protocol: set of messages it can respond to
- Every object inherits methods from `Object`

Protocol for all objects (defined on class `Object`)

`isKindOf: aClass`	Receiver inherits from arg `aClass`?
`isMemberOf: aClass`	Receiver's class is arg `aClass`?
`= anObject`	Equality
`!= anObject`	Inequality
`isNil`	Receiver is nil?
`notNil`	Receiver is not nil?
`print`	Print receiver
`println`	Print receiver, then newline
`error: aSymbol`	Error message
`subclassResponsibility`	Missing method

Example

```
-> (isKindOf: 10 Number)
<True>
-> (isMemberOf: 10 Number)
<False>
-> (class C Object (x))
<class C>
-> (val c (new C))
<C>
-> (isMemberOf: c C)
<True>
-> (notNil c)
<True>
-> (error: c #this-is-error)
run-time error: this-is-error
Method-stack traceback:
  Sent 'error:' in standard input
```

Inheritance

Classes:

- Do not design each object individually:
  create special objects called classes
- Classes are templates for objects called instances

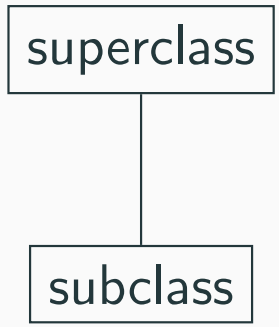
Inheritance:

- Not only lots of instances are similar to each other,
  but lots of classes are similar to each other too
- Exploit similarity between classes using a concept called *inheritance*

"This new class is just like that existing one,
except in the following ways"

Inheritance

- Subclass inherits from the superclass
- Instances of a class understand:
  - Instance methods defined in their class
  - Instance methods defined in their class's superclasses
- A class can have many subclasses
- A subclass can only inherit directly from one superclass



Abstract Class:

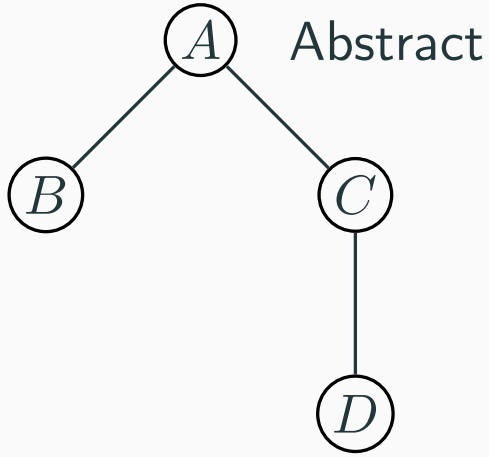
- A class which is designed never to have instances
- Purpose: collect functionalities which other classes will inherit

Inheritance Hierarchy:

- Every class can be both a subclass and a superclass
- This creates a family tree of classes, called the inheritance hierarchy

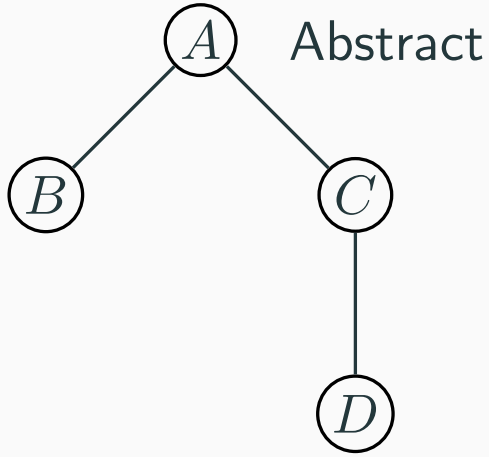
Inheritance Example

```
-> (class A Object () ; abstract
 (method mA () (subclassResponsibility self)))
<class A>
```



Inheritance Example

```
-> (class A Object () ; abstract
 (method mA () (subclassResponsibility self)))
<class A>
```

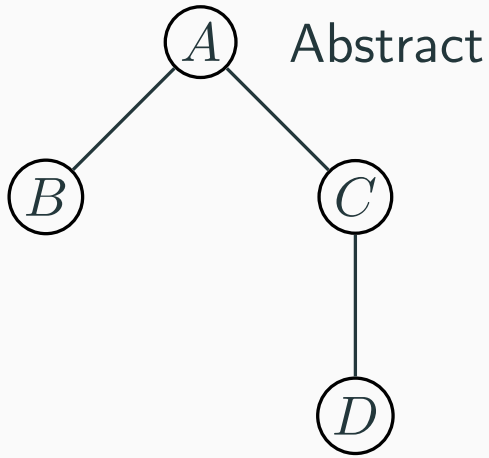



```
-> (val a (new A))
<A>
-> (mA a)
run-time error: subclass failed to implement a method
```

Inheritance Example

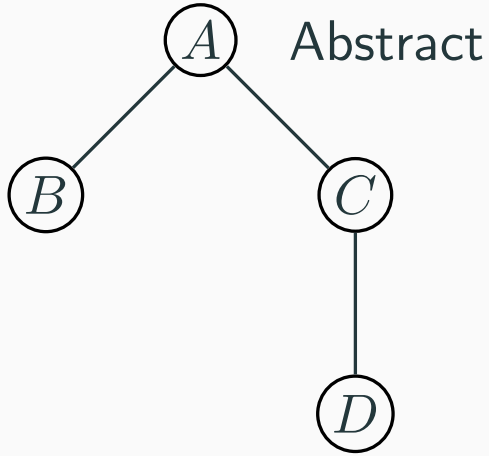
```
-> (class A Object () ; abstract
 (method mA () (subclassResponsibility self)))
<class A>

-> (class B A ()
 (method mA () #mA)
 (method mB () #mB))
<class B>
-> (val b (new B))
<B>
-> (mA b)
mA
-> (mB b)
mB
```



Inheritance Example

```
-> (class A Object () ; abstract
 (method mA () (subclassResponsibility self)))
<class A>
-> (class C A ()
 (method mA () #mA)
 (method mC () #mC))
<class C>
-> (class D C ())
<class D>
-> (val d (new D))
<D>
-> (mA d)
mA
-> (mB d)
run-time error: D does not understand message mB
-> (mC d)
mC
```



Protocol for Class

`new`	answer with a new instance of that class
`protocol`	prints class methods and instance methods
`localProtocol`	print class methods and instance methods that are defined by this class (not messages inherited from superclass)

Protocol for Class

```
-> (class A Object ())
<class A>
-> (protocol A)
(class-method localProtocol primitive ...)
(class-method new primitive ...)
(class-method protocol primitive ...)
(method != (x) ...)
(method = primitive ...)
(method error: primitive ...)
(method isKindOf: primitive ...)
(method isMemberOf: primitive ...)
(method isNil primitive ...)
(method notNil primitive ...)
(method print primitive ...)
(method println () ...)
(method subclassResponsibility primitive ...)
```

Blocks

- Blocks are similar to lambda-expressions
- `(block (formals) expressions)`
- Syntactic sugar for parameterless blocks
  - $\{e_1 \ldots e_n\}$
  - $(\texttt{block}\ ()\ e_1 \ldots e_n)$
- Block is an object like other entities in Smalltalk
- Inside expression is evaluated when the `value` message is sent to it

```
-> (val twice (block (n) (+ n n)))
<Block>
-> (value twice 5)
10
```

Blocks

- Evaluated block returns the result of the last expression

```
-> (val printer (block (item1 item2) (print #item1:)
   (println item1) (print #item2:) (println item2)))
<Block>
-> (value printer 1 #RIT)
item1:1
item2:RIT
RIT
```

Blocks

- Block can access variables declared in enclosing scope
- Variables bindings are determined by the scope where block is created

```
-> (val i 0)
0
-> (val addi (block (x) (+ x i)))
<Block>
-> (value addi 5)
5
-> (set i 9)
9
-> (value addi 2)
11
```

Control Structures

- Smalltalk: the **only** control structure is message passing!
- Control structures are implemented as messages
  that take blocks as arguments
- Conditionals are implemented as messages to Boolean objects
- While loops are implemented as messages to blocks that evaluate to true or false

Conditionals

- Use continuation-passing style to implement conditionals
- Pass two blocks (the continuations) to a Boolean
- `true` continues by executing the first block;
  `false` continues by executing the second block

```
-> (val x 5)
5
-> (val y 10)
10
-> (ifTrue:ifFalse: (< x y) {x} {y})
5
-> (ifTrue:ifFalse: (< x y) x y)
run-time error: SmallInteger does not understand
message value
```

Protocol for Booleans

`ifTrue:ifFalse: trueBlock falseBlock`	Full conditional
`ifTrue: trueBlock`	Partial conditional (for side effect only)
`ifFalse: falseBlock`	Partial conditional (for side effect only)
`not`	Logical negation
`& aBoolean`	Conjunction (and)
`| aBoolean`	Disjunction (or)
`eqv: aBoolean`	Logical equality
`xor: aBoolean`	Logical inequality
`and: aBlock`	Short-circuit conjunction
`or: aBlock`	Short-circuit disjunction

True and False

Message `ifTrue:ifFalse:` always dispatches to one of the following methods:

```
(class True Boolean ()
(method ifTrue:ifFalse: (trueBlock falseBlock)
                          (value trueBlock))
)
(class False Boolean ()
(method ifTrue:ifFalse: (trueBlock falseBlock)
                          (value falseBlock))
)
```

While Loops

```
-> (val n 1)
1
-> (whileTrue: {(< n 10)} {(set n (* n 2))})
nil
-> n
16
```

Protocol for blocks:

`value arguments`	Evaluate, answer value of body
`whileTrue: bodyBlock`	Send value to the receiver, and if answer is true, send value to `bodyBlock` and repeat
`whileFalse: bodyBlock`	Send value to the receiver, and if answer is false, send value to `bodyBlock` and repeat

Summary

This Lecture

- Inheritance, Blocks

Next Lecture

- Initial basis