OBJECT-ORIENTED CONCEPTS, PROJECT WORK

CSSE 120—Rose Hulman Institute of Technology

Exam 2 Facts

- Date: Tuesday, October 16, 2007
- Time: 7:00 to 9:00 PM
- Venue: Section 1 (Delvin) O257
  Section 3 (Curt) O267
  Section 2 (Claude) A-G O257, H-Z O267
- Chapters: Zelle chapters 1 to 12 with greater emphasis on chapters 6 to 12
- Organization: A paper part and a computer part, just as on the first exam. Same resources allowed.

Possible topics for exam 2

- topics for exam 1
- defining functions
- using functions
- decision structures
- exception handling
- loops
  - indefinite(while)
  - interactive
  - sentinel
  - file
  - nested
- computing with Booleans
- random numbers
- top-down design
- bottom-up implementation
- objects
- defining & using new classes
- data processing with Class
- encapsulation
- widgets
- lists (with objects, classes)
- process of OOD
- OO concepts

Object-Oriented Programming

- Technique becoming standard practice in software development
- Facilitates production of complex software
  - More reliable
  - Cost-effective
  - Models real world

Object-Oriented Concepts

- Features that make development truly object-oriented
  - Encapsulation: Separating implementation details of an object from how the object is used
  - Inheritance: Defining new classes to borrow behavior from 1 or more other classes
  - Polymorphism: What an object does in response to a method call depends on the type or class of the object

Encapsulation

- Separates object use (how it is used) from object implementation (what it does)
  - Implementation is independent of how it is used
  - Makes it easier to think about the code
- Client code sees a "black box" with a known interface
- Implementation can change without changing client

Encapsulation Example

Client code

```
g = Fraction(12,6)
h = Fraction(6,11)
print g, h
print g.add(h)
```

Fraction Class

```
class Fraction:
    def __init__(self,
       numerator=0,
       denominator=1):
        …

    def __str__(self):
        …

    def add(self, other):
        …
```

Thinking Inside the Box

Client code

```
g = Fraction(12,6)
h = Fraction(6,11)
print g, h
print g.add(h)
```

```
class Fraction:
    """Without normalization."""
    def __init__(self, numerator=0, denominator=1):
        self.num = numerator
        self.den = denominator

    def __str__(self):
        if self.den == 0:
            return 'undefined fraction'
        fact = gcd(abs(self.num), abs(self.den))
        if self.den < 0:
            fact = -fact
        return str(self.num // fact) + '/' + \
                   str(self.den // fact)

    def add(self, other):
        return Fraction(self.num*other.den + \
                 self.den*other.num, self.den*other.den)
```

Thinking Inside the Box

Client code

```
g = Fraction(12,6)
h = Fraction(6,11)
print g, h
print g.add(h)
```

```
class Fraction:
    """With normalization."""
    def __init__(self, numerator=0, denominator=1):
        if denominator==0:
            self.den = 0
            self.num = 0
        else:
            fact = gcd(abs(numerator), abs(denominator))
            if denominator < 0:
                factor = -factor
            self.num = numerator // fact
            self.den = denominator // fact

    def __str__(self):
        if self.den == 0:
            return 'undefined fraction'
        return str(self.num) + '/' + str(self.den)

    def add(self, other): (unchanged)
```

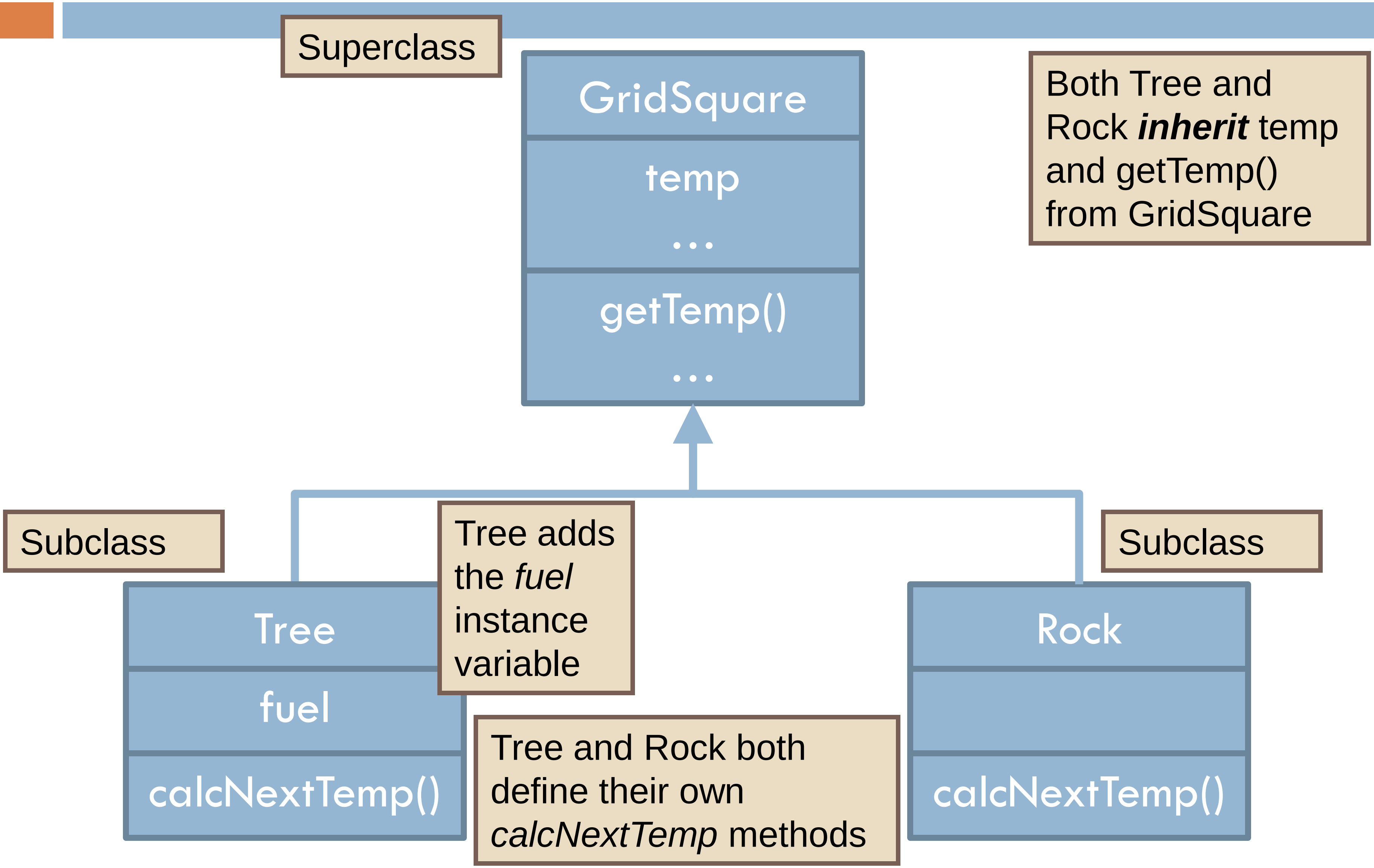
Function vs. object encapsulation

	Functions	Objects
Black box exposes:	Function signature (name, formal parms, return value)	Constructor and method signatures
Encapsulated inside the box (i.e., what we can change without changing client)	Operation implementation	Data storage and operation implementation

Inheritance

- Superclass
  - Base class that new class borrows from
    - Instance variables and methods
  - Models a more general concept
- Subclass
  - New class that borrows behavior from the superclass
  - Models a special case of the more general concept
  - More specialized class that inherits from the superclass
  - Enhances the superclass
  - Is a derived class

Relationship between classes



Superclass

GridSquare

temp

…

getTemp()

…

Both Tree and Rock ***inherit*** temp and getTemp() from GridSquare

Subclass

Tree

fuel

calcNextTemp()

Tree adds the *fuel* instance variable

Subclass

Rock

calcNextTemp()

Tree and Rock both define their own *calcNextTemp* methods

Subclass definition

```
class GridSquare:
    def __init__(self, row, col):
        self.row = row
        self.col = col

class Tree(GridSquare):
    def __init__(self, row, col, fuel):
        GridSquare.__init__(self, row, col)
        self.fuel = fuel
```

Inheritance example

- Using Eclipse, checkout project **OOConcepts** from the svn repository
- Execute the bankAccount program
- Study the code and answer quiz questions 5, 6, and 7

Polymorphism

- Behavior can vary depending on the actual type of an object
- Consider the calcNextTemp() method
  - Both Trees and Rocks can calcNextTemp, but they do so differently
- Consider the ‘+’ operator
  - 5 + 6, 4.3 + 7.0, [1, 2, 3] + [4.3, 7.8]
- Consider Zelle graphics library
  - circle.draw(window)
  - rectangle.draw(window)

A polymorphism example

```
def main():
    animals = [Animal("Garth")]
    animals.append(Cat("Mittens"))
    animals.append(Dog("Blacky"))

    for animal in animals:
        print "\n", str(animal) + " and I " \
              + animal.sound()
```

Look at animalSounds.py in the OOConcepts project

In-class exercise

- Add a CheckingAccount class as a subclass of BankAccount
- Add a transactionCount instance variable to the CheckingAccount class
- Without affecting the superclass BankAccount, enhance the methods deposit() and withdraw() to update transactionCount
- Add method getTransactionCount() to CheckingAccount that returns the transaction count
- Test and commit your work to your SVN repository

Project Milestones

- Session 20 — Program Shows Game State:
  - printBoard() and createBoard(listOfRows)
  - Note that you have to design and implement some data structure to track the board state
- Session 21 — Program Allows Player to Make Any Single Move:
  - makeMove(chooseRow, chooseColumn, placeRow, placeColumn)
- Session 22 — Game Finished
- DATE TBD — Final Presentation

Project Work