Review for the Final Exam

December 8, 2006

ComS 207: Programming I (in Java)
Iowa State University, FALL 2006
Instructor: Alexander Stoytchev

Final Exam

• Time:
  ▪ Tuesday Dec 12 @ 7:00-9:00 p.m.

• Location:
  ▪ same as midterms, i.e., Hoover 2055.

Final Format

• True/False (10 x 1p each = 10p)
• Short answer ( 5 x 3p each = 15p)
• Code Snippets ( 4 x 5p each = 20p)
• What is the output ( 2 x 5p each = 10p)
• Program 1 (20p)
• Program 2 (25p)
• Program 3 (30p)
• TOTAL (130p)

Ways to get an ‘A’ on the Final

• Option 1:
  ▪ 3 program (75p)
  ▪ True/False (10p)
  ▪ Short Answers (15p)
  ▪ TOTAL: (100p)

• Option 2:
  ▪ 2 programs (45p)
  ▪ True/False (10p)
  ▪ Short Answers (15p)
  ▪ Code Snippets (20p)
  ▪ What is the output (10p)
  ▪ TOTAL: (100p)

A Crash Course in Java
Recursive Programming

- Consider the problem of computing the sum of all the numbers between 1 and any positive integer N.
- This problem can be recursively defined as:

\[ \sum_{i=1}^{N} i = N + \sum_{i=1}^{N-1} i \]
\[ = N + (N-1) + \sum_{i=1}^{N-2} i \]
\[ = N + (N-1) + (N-2) + \sum_{i=1}^{N-3} i \]

// This method returns the sum of 1 to num
public int sum (int num)
{
    int result;
    if (num == 1)
        result = 1;
    else
        result = num + sum (n-1);
    return result;
}

Think of recursion as a tree ...

... an upside down tree
Mystery Recursion from HW8

```java
public class Recursion {
    public static void mystery1(int a, int b) {
        if (a <= b) {
            int m = (a + b) / 2;
            System.out.print(m + " ");
            mystery1(a, m-1);
            mystery1(m+1, b);
        }
    }
    public static void main(String[] args) {
        mystery1(0, 5);
    }
}
```

Parameter Passing (primitive types)

- The act of passing an argument takes a copy of a value and stores it in a local variable accessible only to the method which is being called.

```java
int num1=38;
void myMethod(int num2) {
    num2 =50;
    num2 = num2 + num1;
}
```

Before: `num1 38`  num1 = `38`

After: `num1 38`  num1 = `38`
Objects and Reference Variables

Objects (in this case arrays) are also passed by value. In this case, however, the value is the address of the object pointed to by the reference variable.

Before:
```
int[] a={5, 7};
```

After:
```
vvoid myMethod(int[] b) {
    Before: a = 5 7
    b[0]=5;
    myMethod(a);
    After: a = 10 7
}```

The array can be modified through either reference.

Method Overloading

• The compiler determines which method is being invoked by analyzing the parameters

```
float tryMe(int x)
{
    return x + .375;
}

float tryMe(int x, float y)
{
    return x*y;
}
```

Method Overriding

Same Signatures

```
p`````public class Parent {
    public float tryMe(int x) {
        return x + .375;
    }
```

```
```Different Method Bodies

```
p`````public class Child extends Parent {
    public float tryMe(int x) {
        return x*y;
    }
```
Overloading vs. Overriding

- Overloading deals with multiple methods with the same name in the same class, but with different signatures.
- Overriding deals with two methods, one in a parent class and one in a child class, that have the same signature.

Average Example from HW9

```java
public class Average {
    public static double average(double a, double b) {
        return (a+b)/2.0;
    }
    public static double average(double a, double b, double c) {
        return (a+b+c)/3.0;
    }
    public static void main (String[] args) {
        System.out.println (average(1, 2));
        System.out.println (average (1, 2, 3));
    }
}
```

Aggregation Example: Components of a Student

<table>
<thead>
<tr>
<th>Student</th>
<th>John</th>
</tr>
</thead>
<tbody>
<tr>
<td>First Name</td>
<td>John</td>
</tr>
<tr>
<td>Last Name</td>
<td>Smith</td>
</tr>
<tr>
<td>Home Address</td>
<td>21 Jump Street</td>
</tr>
<tr>
<td>School Address</td>
<td>800 Lancaster Ave.</td>
</tr>
</tbody>
</table>
Aggregation

- In the following example, a Student object is composed, in part, of Address objects
- A student has an address (in fact each student has two addresses)
- See StudentBody.java (page 304)
- See Student.java (page 306)
- See Address.java (page 307)
- An aggregation association is shown in a UML class diagram using an open diamond at the aggregate end.

Polymorphism in Nature

Class Hierarchy

```
public abstract class Animal {
    abstract void makeSound();
}

public class Cow extends Animal {
    public void makeSound() {
        System.out.println("Moo-Moo");
    }
}

public class Dog extends Animal {
    public void makeSound() {
        System.out.println("Wuf-Wuf");
    }
}

public class Duck extends Animal {
    public void makeSound() {
        System.out.println("Quack-Quack");
    }
}
```
public class Farm
{
    public static void main(String[] args)
    {
        Cow c = new Cow();
        Dog d = new Dog();
        Duck k = new Duck();
        c.makeSound();
        d.makeSound();
        k.makeSound();
    }
}

Result:
Moo-Moo
Wuf-Wuf
Quack-Quack

public class Farm2
{
    public static void main(String[] args)
    {
        Animal[] a = new Animal[3];
        a[0] = new Cow();
        a[1] = new Dog();
        a[2] = new Duck();
        for (int i = 0; i < a.length; i++)
            a[i].makeSound();
    }
}

Result:
Moo-Moo
Wuf-Wuf
Quack-Quack

public class Farm2b
{
    public static void main(String[] args)
    {
        Animal[] a = new Animal[3];
        a[0] = new Cow();
        a[1] = new Dog();
        a[2] = new Duck();
        for (int i = 0; i < a.length; i++)
            a[i].move();
    }
}

Result:
walk
walk
walk

Not possible since Animal is abstract
Polymorphism via Inheritance

- Now let's look at an example that pays a set of diverse employees using a polymorphic method
  - See Farm.java (page 486)
  - See Staff.java (page 487)
  - See StaffMember.java (page 489)
  - See Volunteer.java (page 491)
  - See Employee.java (page 492)
  - See Executive.java (page 493)
  - See Hourly.java (page 494)

The Animals example with interfaces

In this case Animal is an interface.
public interface Animal {
    public void makeSound();
    public void move();
}

public class Cow implements Animal {
    public void makeSound() {
        System.out.println("Moo-Moo");
    }
    public void move() {
        System.out.println("walk");
    }
}

public class Dog implements Animal {
    public void makeSound() {
        System.out.println("Wuf-Wuf");
    }
    public void move() {
        System.out.println("walk");
    }
}

public class Duck implements Animal {
    public void makeSound() {
        System.out.println("Quack-Quack");
    }
    public void move() {
        System.out.println("fly");
    }
    public void dive() {
        System.out.println("Diving…");
    }
}

Only Ducks can dive.

public class iFarm2 {
    public static void main(String[] args) {
        Animal domestic;
        domestic = new Cow();
        domestic.move();
        domestic = new Dog();
        domestic.move();
        domestic = new Duck();
        domestic.move();
    }
}

Result:
walk  
walk  
fly  

public class iFarm3 {
    public static void main(String[] args) {
        Animal domestic;
        domestic = new Cow();
        //domestic.dive();  // error
        domestic = new Dog();
        //domestic.dive();  // error
        domestic = new Duck();
        //domestic.dive();  // error
        ((Duck)domestic).dive();  // OK, but uses a cast
    }
}

Result:
Ducks can dive.

Exceptions Class Hierarchy

The throw Statement
- Exceptions are thrown using the throw statement
  - Usually a throw statement is executed inside an if statement that evaluates a condition to see if the exception should be thrown
- See CreatingExceptions.java (page 543)
- See OutOfRangeException.java (page 544)
Questions?

THE END