Announcements

- Midterm grades posted

- Program 2 grades
  - Grades posted
  - Honor code reminder

- Program 3 due today

- Program 4 assigned soon
  - Due 6/10
Program 2

<table>
<thead>
<tr>
<th>Operand 1</th>
<th>Operator</th>
<th>Operand 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 2 . 7</td>
<td>+</td>
<td>3 . 3</td>
</tr>
<tr>
<td>0 1 2 3 4 5 6 7 8 9</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Program 2

Assume the String above is in the variable calculation.

```java
int firstSpace = calculation.indexOf(' ');  
int lastSpace = calculation.lastIndexOf(' ');  

double operand1 = Double.parseDouble(calculation.substring(0, firstSpace));  
double operand2 = Double.parseDouble(calculation.substring(lastSpace + 1));  
char operator = calculation.charAt(firstSpace + 1);
```
Now you can determine which calculation to perform using a switch or if/else

double result = 0.0;
switch (operator)
{
    case ‘+’:
        result = operand1 + operand2;
        break;
    case ‘-’:
        result = operand1 - operand2;
        break;
    // and so on
}
double result = 0.0;
switch (operator) 
{
    case '^':
        result = 1.0;
        if(operand2 > 0)
        {
            for (int count = 0; count < (int) operand2; count++)
                result *= operand1;
        }
        else if(operand2 < 0)
        {
            for (int count = 0; count > (int) operand2; count--)
                result *= operand1;
            result = 1.0 / result;
        }
        break;
    case '-':
        result = operand1 - operand2;
        break;
    // and so on
}
Floating-point numbers are imprecise
  ◦ After doing many computations, value may not be exactly the same
  ◦ 5.0 * 3.0 might be different from 1.0 + 2.0 + 3.0 + 4.0 + 5.0

Okay for this assignment for the divide by 0 error checking
JarChecker

- About 1/4 of Program 2 submissions included broken jar files.

- JarChecker now mandatory
  - Beginning with Program 3 (due today)
  - Submitting a broken jar file treated like submitting with a compile error — huge point deduction
Your Program 3 MANIFEST.MF should contain:

Manifest-Version: 1.0
Class-Path: .
Main-Class: UNCStats

If not, FIX IT.
Cheating and the Honor Code

- Honor code violations *include*:
  - Sharing code in any way
  - Discussing your algorithms or solutions

- Read the CS honor code AGAIN.
  - Any suspicious submissions will be reported to honor board.

- You are also subject to the UNC honor code
  - Both are linked to from the course syllabus.
Cheating and the Honor Code

- Very easy to prove cheating on programs

- If found guilty...
  - ONE YEAR SUSPENSION
  - An F in this class and any other classes you are taking

- We take the honor code seriously

- Just don’t cheat.
Questions?
Today in COMP 110

- A couple of notes
- Constructors
- Static Methods
Make sure you are running the class that has a main method in it

Otherwise, you will get this:

java.lang.NoSuchMethodError: main
Exception in thread "main"
Local variables

What does the greet() method output?

```java
public class Example {
    private String str = "hello";
    public void doStuff() {
        String str = "goodbye";
    }
    public void greet() {
        doStuff();
        System.out.println(str);
    }
}
```

- Outputs hello. Why?
- `doStuff()` uses local variable `str`, not instance variable `str`
Local variables

What does the greet() method output?

```java
public class Example {
    private String str = "hello";
    public void doStuff() {
        str = "goodbye";
    }
    public void greet() {
        doStuff();
        System.out.println(str);
    }
}
```

- Outputs goodbye. Why?
- doStuff() uses instance variable str
public class AnotherExample {
    private Student jack;

    public void myMethod() {
        jack = new Student();
        jack.setName("Jack Smith");
        jack.setAge(19);
    }
}
Create and initialize new objects

Special methods that are called when creating a new object

Student jack = new Student();

Calling a constructor
Create an object jack of class Student
Student jack = new Student();

Scanner keyboard = new Scanner(System.in);
Create an object by calling a constructor

Assign memory address of object to variable
Return memory address of object

Create an object keyboard of class Scanner
 Constructors

- Can perform any action you write into a constructor’s definition

- Meant to perform initializing actions
  - For example, initializing values of instance variables
Similar to mutator methods

- However, constructors create an object in addition to initializing it.
- Like methods, constructors can have parameters.
public class Pet {
  private String name;
  private int age;
  private double weight;

  public Pet() {
    name = "No name yet.";
    age = 0;
    weight = 0;
  }

  public Pet(String initName, int initAge, double initWeight) {
    name = initName;
    age = initAge;
    weight = initWeight;
  }
}
public void setPet(String newName, int newAge, double newWeight) {
    name = newName;
    age = newAge;
    weight = newWeight;
}
public Pet(String initName, int initAge, double initWeight) {
    name = initName;
    age = initAge;
    weight = initWeight;
}

Same name as class name

No return type

Parameters

Body
Initializing instance variables

- Constructors give values to all instance variables

- Even if you do not explicitly give an instance variable a value in your constructor, Java will give it a default value

- Normal programming practice to give values to all instance variables
Default constructor

- Constructor that takes no parameters

```java
public Pet()
{
    name = "No name yet.";
    age = 0;
    weight = 0;
}
```

- Java automatically defines a default constructor if you do not define any constructors
Default constructor

- If you define at least one constructor, a default constructor will not be created for you.
Several constructors

- You can have several constructors per class
  - They all have the same name, just different parameters
public class Pet {
    private String name;
    private int age;
    private double weight;

    public Pet() {
        name = "No name yet.";
        age = 0;
        weight = 0;
    }

    public Pet(String initName, int initAge, double initWeight) {
        name = initName;
        age = initAge;
        weight = initWeight;
    }
}
Calling a constructor

Pet myPet;
myPet = new Pet("Smokey", 15, 65);

- You cannot use an existing object to call a constructor:

myPet.Pet("Fang", 3, 155.5); // invalid!
myPet.setPet("Fang", 1, 155.5);
Calling methods from constructors

- Just like calling methods from methods

```java
public Pet(String initName, int initAge, double initWeight)
{
    setPet(initName, initAge, initWeight);
}

public void setPet(String newName, int newAge,
                    double newWeight)
{
    name = newName;
    age = newAge;
    weight = newWeight;
}
```
Calling methods from constructors

- Can cause problems when calling public methods
  - Problem has to do with inheritance, chapter 8
  - Another class can alter the behavior of public methods

- Can solve problem by making any method that constructor calls \texttt{private}
Example: Pet class

```java
public class Pet {
    private String name;
    private int age;
    private double weight;

    public Pet(String initName, int initAge, double initWeight) {
        set(initName, initAge, initWeight);
    }

    public void setPet(String newName, int newAge, double newWeight) {
        set(newName, newAge, newWeight);
    }

    private void set(String newName, int newAge, double newWeight) {
        name = newName;
        age = newAge;
        weight = newWeight;
    }
}
```
Self-test Questions

- If a class is named Student, what name can you use for a constructor of this class?
- What return type do you specify for a constructor?
- What is a default constructor?
Instance variables

private int age;
private String name;

Methods

public int getAge()
{
    return age;
}

Calling methods on objects

Student std = new Student();
std.setAge(20);
System.out.println(std.getAge());
static

- static variables and methods belong to a class as a whole, not to an individual object

- Where have we seen static before?

- When would you want a method that does not need an object to be called?
// Returns x raised to the yth power, where y >= 0.
public int pow(int x, int y)
{
    int result = 1;
    for (int i = 0; i < y; i++)
    {
        result *= x;
    }
    return result;
}
static, some examples

- static constants and variables
  - private static final int FACE_DIAMETER = 200;
  - public static final int FEET_PER_YARD = 3;
  - private static int numberOfInvocations;

- static methods
  - public static void main(String[] args)
  - public static int pow(int x, int y)
public class MathUtilities {
    // Returns x raised to the yth power, where y >= 0.
    public static int pow(int x, int y) {
        int result = 1;
        for (int i = 0; i < y; i++) {
            result *= x;
        }
        return result;
    }
}
Static variables and methods can be accessed using the class name itself:
- `DimensionConverter.FEET_PER_YARD`
- `int z = MathUtilities.pow(2, 4);`
public class SomeClass
{
    public static final double PI = 3.14159;
    private boolean sunny = true;

    public static double area(double radius)
    {
        sunny = false;
        return PI * (radius * radius);
    }
}

- Code will not compile
- static methods are invoked without an object
  - no access to instance variables or non–static methods
public class SomeClass
{
    public static final double PI = 3.14159;
    public int data = 12;

    private void printData()
    {
        System.out.println(data);
    }

    public static double area(double radius)
    {
        printData();          // ERROR!
        return PI * (radius * radius);
    }
}
public class SomeClass
{
    public static final double PI = 3.14159;

    private void printPi()
    {
        System.out.println(PI);
        System.out.println(area(3.0));
    }

    public static double area(double radius)
    {
        return PI * (radius * radius);
    }
}

- Nonstatic methods CAN call static methods and access static variables
public class SomeClass
{
    public static final double PI = 3.14159;

    private void printPi()
    {
        System.out.println(PI);
    }

    public static double area(double radius)
    {
        SomeClass sc = new SomeClass();
        sc.printPi();
        return PI * (radius * radius);
    }
}
Self-test questions

- Can you call a nonstatic method from a static method?
- Can you call a static method from a nonstatic method?
- Can you access an instance variable inside a static method?
import java.util.*;
public class MyClass {
    public static void main(String[] args) {
        System.out.println("Give me a number, and I will " +
                          "tell you its square and its square’s square.");
        Scanner kb = new Scanner(System.in);
        int num = kb.nextInt();
        int numSquared = num * num;
        System.out.println("The square is "+numSquared);
        int numSquaredSquared = numSquared * numSquared;
        System.out.println("The square’s square is "+
                           numSquaredSquared);
    }
}
import java.util.*;
public class MyClass {
    public static int square(int x) {
        return x * x;
    }

    public static void main(String[] args) {
        System.out.println("Give me a number, and I will "+"tell you its square and its square’s square.");
        Scanner kb = new Scanner(System.in);
        int num = kb.nextInt();
        int numSquared = square(num);
        System.out.println("The square is "+numSquared);
        int numSquaredSquared = square(numSquared);
        System.out.println("The square’s square is "+numSquaredSquared);
    }
}
The Math class

- Provides many standard mathematical methods, all static
  - Do not create an instance of the Math class to use its methods
- Call Math class’ methods using class name
  - Math.abs
  - Math.max
  - Math.min
  - Math.pow
  - Math.round
  - Others
- Predefined constants
  - Math.PI
  - Math.E
public static double largeToSmallthPower(int a, int b)
{
    double small = Math.min(a, b);
    double large = Math.max(a, b);
    return Math.pow(large, small);
}

public static double area(double radius)
{
    return Math.PI * (radius * radius);
}
Math.round

- Math.round: returns closest long (or int, if using a float) to argument
  - Math.round(2.3)
    - Returns 2
  - Math.round(2.7)
    - Returns 3
Math.floor: returns largest \textit{double} value less than or equal to argument and equal to a mathematical integer

- Math.floor(2.3)
  - Returns 2.0
- Math.floor(2.7)
  - Returns 2.0
Math.ceil

- Math.ceil: returns smallest **double** value greater than or equal to argument and equal to a mathematical integer
  - Math.ceil(2.3)
    - Returns 3.0
  - Math.ceil(2.7)
    - Returns 3.0
What if you want an int?

- Math.ceil returns a double
  - Math.ceil(5.6) returns 6.0

```java
int num = (int) Math.ceil(5.6);
```
Wrapper classes

- All primitive types have an associated wrapper class
- We have seen some of these before (where?):
  - Byte
  - Short
  - Integer
  - Long
  - Float
  - Double
  - Character
  - Boolean
int num = Integer.parseInt("36");

- Integer.parseInt, Double.parseDouble, etc. are all static methods

- These wrapper classes also have nonstatic methods
Wrapper classes

- You can create an instance of a wrapper class and use the instance to convert the value to different types.

Example:
- `Integer num = new Integer(36);`
- `double numAsDouble = num.doubleValue();`
Wrapper classes

- Useful **static** constants and methods
- Examples:
  - `Integer.MAX_VALUE`
  - `Double.MIN_VALUE`
  - `Float.parseFloat("23.7");`
  - `Long.toString(368);`
class Character

- Character.toUpperCase
- Character.toLowerCase
- Character.isUpperCase
- Character.isLowerCase
- Character.isWhitespace
- Character.isLetter
- Character.isDigit