More Methods

GEEN163

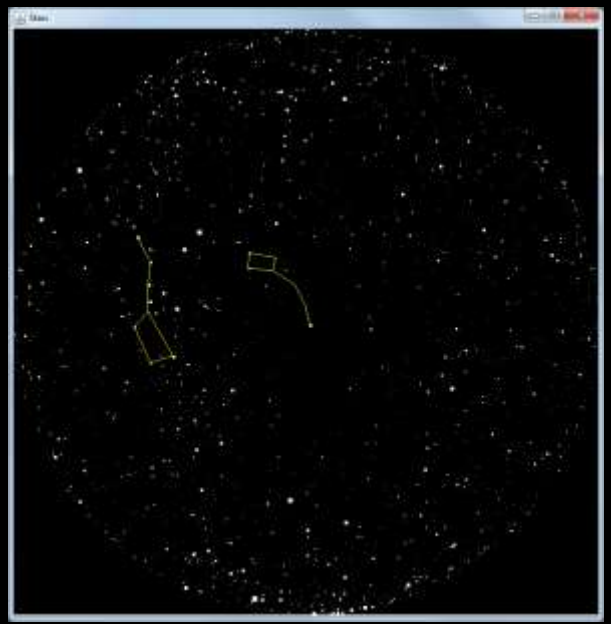
"There is ***method*** *to her madness."*

TuringsCraft

- Read the material in chapter 7 of the textbook
- Answer the questions in section 7 of the TuringsCraft tutorial
  - 4 points for each correct answer
  - 100 point maximum
- Due by midnight on Thursday (tomorrow)

Programming Homework

- A new programming assignment has been posted on Blackboard
- This program draws a picture of the night sky
- You are required to write a method



Chancellor's Forum

- The last Chancellor's Forum for students this academic year will be held from 5:30 – 7:30 p.m., on Thursday, March 17, in the Academic Classroom Building, room 101

- Chancellor's Forums are designed to provide valuable information about university initiatives and strategic direction. Students are encouraged to attend the forums

Constructors are Usually Simple

- Most constructor methods simply copy the parameter to a class variable
- Some constructors set class variables to a constant

- Some constructors are more complex. The constructor for Scanner may have to check on a network connection

Multiple Constructors

- A class may have more than one constructor
- Different constructors must have a different number or type of parameters
- All constructors must have the same name as the class

- This is known as ***polymorphism***

Two Constructors for Line

```
public class Line {
   double slope;                  // slope of the line
   double intercept;              // Y intercept of the line

   public Line(double tilt) { // constructor
          slope = tilt;
          intercept = 0.0;
   }
  public Line(double tilt, double y) { // constructor
          slope = tilt;
          intercept = y;
   }
}
```

Polymorphism Requires Different Parameters

- You cannot have the following two methods because they both have one double parameter

```
public double half(double dog) {    // Same as below
        return dog/2.0;
  }

 public double half(double cat) {   // Same as above
         return cat * 0.5;
 }
```

Which statement calls the constructor?

A. Line cheetah = new Line(55.0);
B. cheetah = Line(66.0);
C. Line(cheetah, 77.0);
D. new cheetah = Line(44.0);
E. none of the above

Getters and Setters

- Some methods are very simple and only return or set an instance variable

```
int wolf;              // instance variable
int getWolf() {
    return wolf;
}
void setWolf(int lupus) {
     wolf = lupus;
}
```

Accessor or Getter Methods

- Accessor methods return some value from the object
- Accessor methods do not usually change anything in the object
- Getter methods usually have no parameters and return a value of the same type as the instance variable
- Examples of accessor methods include:
  - String **`length()`** method

Modifier or Setter Methods

- Modifier methods change the state of an object
- A modifier method may just set the value of a single variable or may perform a lengthy sequence of actions
- Setter methods are often void methods and usually have one parameter

Example Class

```
public class Widget {
  private int count = 0;                    // class data value

  public Widget( int num ) {                // constructor
    count = num;
  }
  public void inc() {                       // method to increment
    count++;
  }
  public int getCount() {                   // accessor method
    return count;
  }
  public void setCount( int value ) {       // modifier method
    count = value;
  }
}
```

Naming Convention

- Constructors must have the same name as the class
- Class names start with an uppercase letter
- Objects are written in lowercase
- Method names are in lowercase
- Accessor methods have names that can be used as nouns or getVariable
- Modifier methods have names that can be used as verbs or setVariable

Write with your team

- Given the following class, write get and set methods for numMembers

```
public class TeamWork {
        private int numMembers;
        private String name;
}
```

Possible Solution

```
public class TeamWork {
        private int numMembers;
        private String name;

        public int getNumMembers() {
                return numMembers;
        }
        public void setNumMembers( int count ) {
                numMembers = count;
        }
}
```

Calling Methods

- In Java you can use a method of an object by writing the object's name, a period, the name of the method.

```
Widget thing = new Widget();
thing.whatever( 5 );
```

- This calls the whatever method of Widget object thing passing it an integer 5 as a parameter

Inheritance

- You can extend an existing class to add new features
- We used inheritance to extend JApplet
- The new class has all of the data values and methods of the parent classes

Inheritance Example

- Some of our programs have input numbers from the JTextField
- The getText method of JTextField only returns a string
- We can extend JTextField to add methods to get a double or an int

Extending JTextField

```
public class NumField extends
                        javax.swing.JTextField {
    public int getInt() {
        String text = getText();
        int number = Integer.parseInt(text);
        return number;
    }

    public double getDouble() {
        String text = getText();
        double number = Double.parseDouble(text);
        return number;
    }
}
```

Using NumField

```
// class variables at the beginning
NumField inYear = new NumField();
NumField inLoan = new NumField();
            ...

public void actionPerformed(
            java.awt.event.ActionEvent thing) {
  double loan = inLoan.getDouble();
  double m = inYear.getDouble() * 12.0;
  double pay = loan * etc.;
  answer.setText("Monthly payments of $"+pay);
}
```

Parent Method Available

- You do not need the Java source code of a class to inherit from it
- You can extend a system class
- All of the original methods are available

```
NumField inYear = new NumField();
inYear.setText("default");
```

Little Programs

- Methods are small programs
- A method often has parameters
  - Parameters are data given to the method
  - Most methods do **NOT** read from the keyboard
  - Most methods do **NOT** display on the screen
- Methods usually return one value
- Methods are very useful to avoid writing the same code many times with different variables

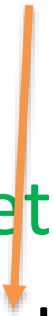
Separate Names

- The names in a method have no connection to the names in the main program

Parameters Are Copied to Method

```
// Main program
int bird = 7, eagle;
eagle = twice( bird );          // eagle becomes 14


// The twice method
int twice( int turtle ) {       // turtle is 7
        int tortoise = 2 * turtle;
        return tortoise;
}
```



Two Kinds of Returns

Value-Returning

Always returns a single value to its caller and is called from within an expression.

Void

Never returns a value to its caller, and is called as a separate statement.

A void Method Call

```
public class VoidMeth {
    public static void main(String[] args) {
      displayMessage( 'A' ) ;   //void method call
      System.out.println("Good Bye");
    }

    static void  displayMessage( char grade) {
      System.out.println("I want an "+ grade );
    }
}
```

This program displays

```
I want an A
Good Bye
```

`return` Statement

- All methods (except void methods and constructors) must return a value
- The return statement contains the value that is given back to the main program
- The return statement is often the last statement in the method

```
return dog;
return 47;
return cat + mouse;
```

Running off the end

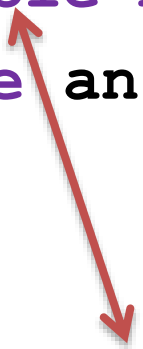
- When a void method runs past the last line of the method, it will return to the calling program
- A void method may use a return statement, but without any value on it

```
return ;
```

Return Type

- The type of the return value is specified in the method header.
- The type in the header must match the type in the return statement

```
public double myMethod(int pig, float goat) {
      double answer = … ;
                                  // method body here
      return answer ;
}
```



Type must match

Methods in Equations

- A method that returns a value is normally used in an equation

```
int rtnMeth( double dog ) { // method definition
        return (int)dog + 2;
}

int cat = 3 * rtnMeth( 5.75 );  // using the method
```

Return Value in an Expression

```
int pig = 5, hog = 3, sow;
sow = square( pig ) + hog;

int square( int cat ) {
    return cat * cat;
}
```

Return Value in an Expression

```
int pig = 5, hog = 3, sow;
sow = square(  5  ) + hog;

int square( int cat ) {
    return cat * cat;
}
```

Return Value in an Expression

```
int pig = 5, hog = 3, sow;
sow = square(  5  ) + hog;

int square( int  5  ) {
    return cat * cat;
}
```

Return Value in an Expression

```
int pig = 5, hog = 3, sow;
sow = square(  5  ) + hog;

int square( int  5  ) {
    return  5  *  5 ;
}
```

Return Value in an Expression

```
int pig = 5, hog = 3, sow;
sow = square(  5  ) + hog;

int square( int  5  ) {
    return  25 ;
}
```

Return Value in an Expression

```
int pig = 5, hog = 3, sow;
sow =         25        + hog;

int square( int   5   ) {
    return  25 ;
}
```

Return Value in an Expression

```
int pig = 5, hog = 3, sow;
sow =         25        +  3 ;

int square( int  5  ) {
    return  25 ;
}
```

Return Value in an Expression

```
int pig = 5, hog = 3, sow;
sow =          28          ;

int square( int  5  ) {
    return  25 ;
}
```

What is displayed?

```
public static void main(String[] cow) {
   int bunny = 7, hare;
   hare = what( bunny) + 2;
   System.out.println( hare );
}
int what( int rat ) {
   int mouse = rat – 3;
   return mouse;
}
```

A. 2

B. 3

C. 4

D. 6

E. 7

Example Method

- Assume you have a number between -1 and +1 and you want to scale it to be between 0 and 500
- Consider a method that takes a double parameter between -1.0 and 1.0 and returns an int between 0 and 500

Method Header

- The parameter is double and the return type is int

```
public int convert( double x )
```

- It is useful to write the header of a method to help understand exactly what is going in to the method and what is being returned

Adding the Body

- The parameters go from -1.0 to 1.0 or a range of 2.0 and the output range is 500
- Negative values of the parameter will be less than 250 and positive more than 250

```
public int convert( double x ) {
    return (int)(x * 250.0) + 250;
}
```

Calling a Method in the same Class

- A call to a method in the same class needs the method name followed by the arguments

```
myMethod( a, b );
```

Calling a Method in Another Class

- A call to a **static** method in the another class needs the class name, a period, the method name followed by the arguments

```
YourClass.myStaticMethod( a, b );
```

- Math and JOptionPane are examples of classes with static methods

Calling a Method in Another Class

- A call to a **non-static** method in the another class needs an object's variable name, a period, the method name followed by the arguments.

```
YourClass myObject = new YourClass();
myObject.myDynamicMethod( a, b );
```

- The Scanner class has non-static methods

What is displayed by this program?

```
int deer = 2, bear = 7;          // start here
Geen163 myObj = new Geen163();
myObj.doWhat( deer, bear);
```

. . . ***in class Geen163*** . . .

```
void doWhat( int goat, int pig) {
  goat = goat + 3;
  System.out.println("goat="+goat+" pig="+pig);
}
```

A. goat=2 pig=7

B. goat=5 pig=7

C. goat=7 pig=5

D. none of the above

Write a method to average two numbers

- Write a simple method (ignore any class definition for now) that averages two double numbers

Write a method to average two numbers

- The method will need two parameters of type double
- The return value of the method will be the average and should be a double

A method to average two numbers

```
double average( double num1, double num2) {
        return (num1 + num2) / 2.0;
}
```

A method to average two numbers

```
double average( double num1, double num2) {
        double avg = (num1 + num2) / 2.0;
        return avg;
}
```

Incorrect

```
double average( double num1, double num2) {
        num1 = keyboard.nextDouble();
        num2 = keyboard.nextDouble();
        double avg = (num1 + num2) / 2.0;
        System.out.println("avg = " + avg );
        return avg;
}
```

Methods calling Methods

```
int dog = 5, cat;
cat = methA( dog );
```

```
int methA( int cow ) {                    // cow = 5
   int bull =  2 * cow;
   steer = methB( bull )
   return steer
}
```

```
int methB( int lizard ) {
   return lizard + 3;
}
```

Methods calling Methods

```
int dog = 5, cat;
cat = methA( dog );
```

```
int methA( int cow ) {              // cow = 5
   int bull =  2 * cow;             // bull = 10
   steer = methB( bull )
   return steer
}
```

```
int methB( int lizard ) {
   return lizard + 3;
}
```

Methods calling Methods

```
int dog = 5, cat;
cat = methA( dog );
```

```
int methA( int cow ) {          // cow = 5
   int bull =  2 * cow;         // bull = 10
   steer = methB( bull )
   return steer
}
```

```
int methB( int lizard ) {       // lizard = 10
   return lizard + 3;
}
```

Methods calling Methods

```
int dog = 5, cat;
cat = methA( dog );
```

```
int methA( int cow ) {          // cow = 5
   int bull =  2 * cow;         // bull = 10
   steer = methB( bull )
   return steer
}
```

```
int methB( int lizard ) {       // lizard = 10
   return lizard + 3;           // returns 13
}
```

Methods calling Methods

```
int dog = 5, cat;
cat = methA( dog );
```

```
int methA( int cow ) {          // cow = 5
  int bull =  2 * cow;          // bull = 10
  steer = methB( bull )         // steer 13
  return steer
}
```

```
int methB( int lizard ) {       // lizard = 10
  return lizard + 3;            // returns 13
}
```

Methods calling Methods

```
int dog = 5, cat;
cat = methA( dog );
```

```
int methA( int cow ) {          // cow = 5
  int bull =  2 * cow;          // bull = 10
  steer = methB( bull )         // steer 13
  return steer                  // returns 13
}
```

```
int methB( int lizard ) {       // lizard = 10
  return lizard + 3;            // returns 13
}
```

Methods calling Methods

```
int dog = 5, cat;
cat = methA( dog );                    // cat = 13
```

```
int methA( int cow ) {                 // cow = 5
   int bull =  2 * cow;                // bull = 10
   steer = methB( bull )               // steer 13
   return steer                        // returns 13
}
```

```
int methB( int lizard ) {              // lizard = 10
   return lizard + 3;                  // returns 13
}
```

What is displayed?

```
int methA(int dog) {
   int cat = methB( dog + 1 );
   return cat;
}
int methB(int goat) {
   return goat * 2;
}
public static void main(String[] x) {
   int fish = methA( 4 );
   System.out.println( fish );
}
```

A. 4

B. 5

C. 10

D. none of the above

Changing the Parameters Does **Not** Change the Calling Arguments

- If a method changes the value of a parameter variable, it does not change the value of the variable in the calling program.
- The values of the argument variables are copied to the method parameters when the method is called, but are **not** copied back when the method returns.

Parameter Values Not Changed

```
public class ModParm {
     public static void main(String[] args ){
        int   a = 5, b = 47, c = -1;
        System.out.println("main a="+a+"  b="+b+" c="+c);
        c = example( a, b );
        System.out.println("main a="+a+"  b="+b+" c="+c);
     }
     static int example( int x, int y ) {
         x = 13;      // parameter changed
         System.out.println("example x="+x+" y="+y);
        return x + y;
     }
  }
```

Output

```
main a=5  b=47  c=-1
example x=13 y=47
main a=5  b=47  c=60
```

What is printed by this program?

```
int deer = 2, bear = 7;      // start here
Geen163 myObj = new Geen163();
myObj.doWhat( deer, bear);
 System.out.println(" deer="+deer);
```

. . . in class Geen163 . . .

```
void doWhat( int goat, int pig) {
   goat = goat + 3;
   System.out.print("goat="+goat);
}
```

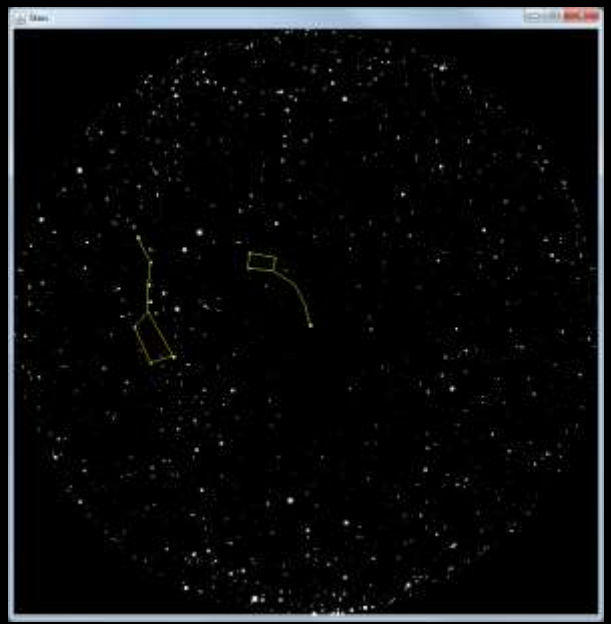
A. goat=2 deer=7
B. goat=5 deer=2
C. goat=5 deer=5
D. none of the above

TuringsCraft

- Read the material in chapter 7 of the textbook
- Answer the questions in section 7 of the TuringsCraft tutorial
  - 4 points for each correct answer
  - 100 point maximum
- Due by midnight on Thursday (tomorrow)

Programming Homework

- A new programming assignment has been posted on Blackboard
- This program draws a picture of the night sky
- You are required to write a method



Chancellor's Forum

- The last Chancellor's Forum for students this academic year will be held from 5:30 – 7:30 p.m., on Thursday, March 17, in the Academic Classroom Building, room 101

- Chancellor's Forums are designed to provide valuable information about university initiatives and strategic direction. Students are encouraged to attend the forums