

# Java Classes and Methods

GEEN163 Introduction to Computer  
Programming

“Our object in the construction of the state is the greatest happiness of the whole, and not that of any one **class.**”

Plato

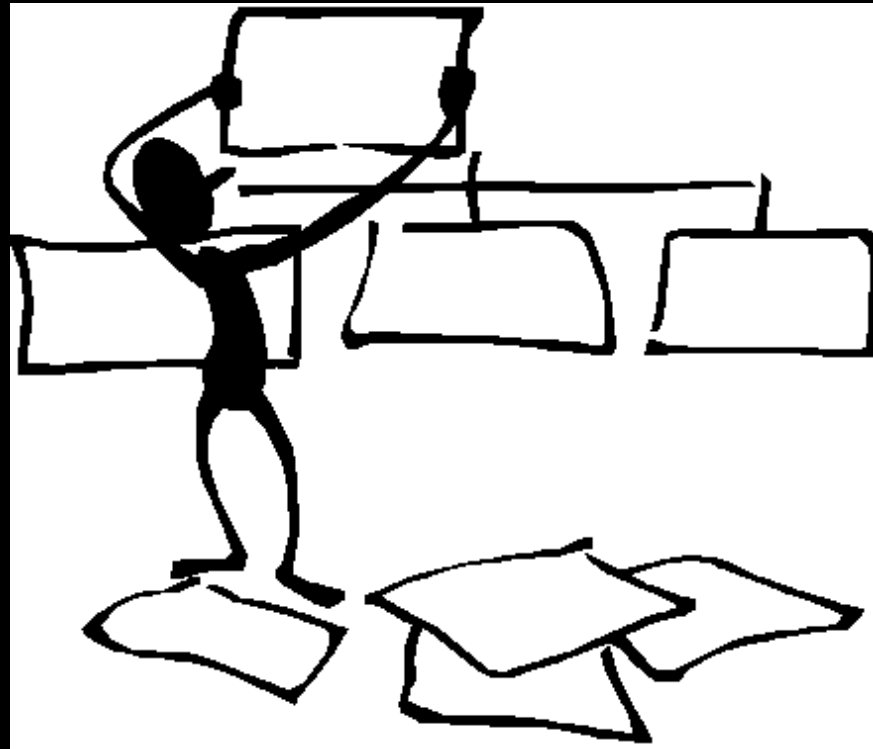
# TuringsCraft

- 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 Monday, April 1

# 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

# TA Reorganization



Luis Cabrera will be the recitation instructor on Thursdays & Fridays replacing Micayla Goodrum

# Parameters Must Match Type

- Method

```
double myfunc( int cat, String dog) {  
  ... }  
}
```

- Calling program

```
int cow;  
String bull;  
double goat;  
goat = myfunc( cow, bull );
```



# Variable Type

- In the method header, you need to specify the variable type

```
int myFunc( int trout, double salmon)
```

- When you call the method, you do not need to specify the type of the parameters

```
int fish, cow = 47;
```

```
fish = myFunc( cow, 3.13 );
```

# Argument Values Copied

- When a method call is executed, the values of the argument variables (or constants) are copied to the parameter variables.
- The method uses a copy of the argument variables.



# Parameter Values Copied

```
public class PassParm {
    public static void main(String[] args ) {
        int cat = 5, bull = 7, dog = 1;
        dog = doIt( cat, bull );
        System.out.println("main "+dog);
    }

    static int doIt( int ant, int bug ) {
        System.out.println("doIt"+ant+" "+bug);
        return ant + bug;
    }
}

    Displays:      doIt 5 7
                  main 12
```

# Write a method with your team

- Write a method that takes a double parameter and returns the square of that number

# Possible Solution

- Write a method called square that takes a double parameter and returns the square of that number

```
double square( double num ) {  
    return num * num;  
}
```

# Accessing Object Fields

- The field variables of an object can be used by any method of that object just like a variable defined in the method.

```
public class Student {  
    int        identifier;  
    public double    grade;  
    public String    name;  
    public void setGrade(double score) {  
        grade = score;  
    }  
}
```

# What is displayed?

```
int cat = 7, dog = 2;  
dog = tryit( cat );  
System.out.println("main "+dog);
```

- - - - -

```
int tryit( int cow ) {  
    int goat = cow + 3;  
    System.out.print("tryit "+ goat );  
    return goat;  
}
```

- A. tryit 10 main 10
- B. main 10 tryit 10
- C. tryit 10 main 7
- D. main 2 tryit 10

# Access Outside the Class

- Public field variables can be accessed from outside of the class.
- To access a field variable, write the object name, a period and then the field variable's name

```
Student fred = new Student();
```

```
fred.name = "Fred Smith";
```

# Access Rights

Data values or methods can be used if specified:

- **public** – anywhere
- **private** – only in that class
- **protected** – only in that class or any class that extends it
- *nothing* – only in that package

# Private example

```
public class Student {  
    public int        id;  
    private double    grade;  
    public String     name;  
    public setGrade(double score) {  
        grade = score;  
    }  
    public double getGrade() {  
        return grade;  
    }  
}
```



# Accessing the Example

- In another class, you can access the public but not the private values

```
public class myProg {  
    public static void main(String[] x ) {  
        Student fred = new Student();  
        fred.name = "Sally";  
        fred.grade = 4.0; // not allowed  
        fred.setGrade( 4.0 ); // allowed  
    }  
}
```

# Data Abstraction

- Classes are described by their interface, that is, what they can do and how they are used.
- The internal data and operation are hidden.
- In this way if the internal operation is changed, programs using the class will not be impacted as long as the interface remains consistent.

```
public class Student {  
    public int id;  
    public double getIdPlus() {  
        return id + 5;  
    }  
}
```

*// in another program*

```
Student dog = new Student();  
dog.id = 3;  
int cat = dog.getIdPlus();
```

## What is cat?

- A. 3
- B. 5
- C. 8
- D. Error

# Method Purpose

- Constructors
  - Initialize an object
- Modifiers
  - Change the values of an object
- Accessors
  - Return the value of a object without changing anything
- Function
  - Compute some value or perform some action

# Constructors

- A constructor method is automatically called when an object is created
- The name of the constructor method is always the same as the class name
- Constructors can be used to initialize the values of an object's variables
- Constructors may or may not have parameters
- Constructors do not have a return type. They are not void

# Constructor Example

```
public class Widget {  
    private int count = 0;  
    /* constructor method */  
    public Widget( int aardvark) {  
        count = aardvark;  
    }  
}
```

# Using Constructors

- The constructor method is called when you create an object

```
Widget gorilla;
```

```
gorilla = new Widget( 5 ); // constructor
```

# Tryit

```
public class Rodent{  
    double rat;  
    int mouse;
```

```
// Write a constructor method to initialize  
// the class variables
```

```
}
```

```
// in another program using the Rodent class
```

```
Rodent mole = new Rodent( 5.6, 14 );
```

```
Rodent gerbil = new Rodent( 3.25, 8 );
```



# Possible Constructor

```
public class Rodent{  
    double rat;  
    int mouse;  
    public Rodent( double vole, int shrew ) {  
        rat = vole;  
        mouse = shrew;  
    }  
}
```

// in a program using the Rodent class

```
Rodent mole = new Rodent( 5.6, 14 );
```

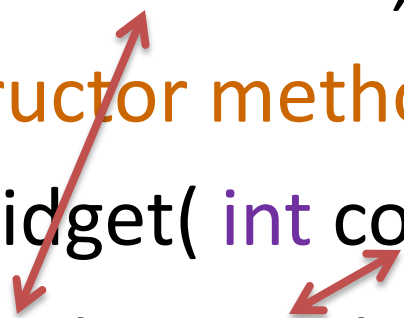
```
Rodent gerbil = new Rodent( 3.25, 8 );
```

# this

- The Java keyword “**this**” means this object
- You can use **this** to access anything of the object, but not the class
- If you have a field named **xyz**, then you can access the field as **this.xyz**

# Another Constructor Example

```
public class Widget {  
    private int count = 0;           // field variable  
    /* constructor method */  
    public Widget( int count ) {    // parameter  
        this.count = count;  
    }  
}
```



# 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 Line Constructors

```
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;  
    }  
}
```

# Accessor Methods

- Accessor methods return some value from the object
- Accessor methods do not usually change anything in the object
- Examples of accessor methods include:
  - String **length()** method

# Modifier 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
- Modifier methods are often void methods



# 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;  
    }  
}
```

# What is displayed?

```
Widget cow = new Widget( 3 );  
cow.setCount( 7 );  
cow.inc();  
int bull = cow.getCount();  
System.out.println( bull );
```

A. 3

B. 4

C. 7

D. 8

# Naming Convention

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

# 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");
```

# No Thursday lab at 10:00

- The Fall Convocation is Thursday
- All classes from 10:00 to noon on Thursday will be cancelled
- The 8:00 Thursday lab will be held as usual

# 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 5:00pm on Monday, April 1

# 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