CSE 142

Methods

Overview

- Quick Review
  - Defining classes, instance variables, and constructors
- Today
  - Defining methods
  - Parameters
  - Method execution
  - Introduction to Debugging
  - Comments and Documentation
- Reading
  - Dugan notes: Ch. 7, all of Ch. 9
  - Niño & Hosch: Sec. 5.2

Drawing House Objects

- Want to be able to send a House object a message, e.g. addTo:

```
House h = new House( );
GWindow w = new GWindow( );
h.addTo(w);
```

- But House objects don't yet know how to respond to the addTo message!
- We have to write an addTo *method* (or *function*, or *procedure*) in the House class that defines how Houses respond to addTo.
  - How? By adding the House's frame and roof to a GWindow!

A Method

- Inside the House class in House.java:

```
/** Add this House to a drawing window
 * @param gw The window the house should be displayed on */
public void addTo(GWindow gw) {
    this.frame.addTo(gw);
    this.roof.addTo(gw);
}
```

- Pattern, if message has one argument and no result:

```
/** comment explaining method and its parameters */
public void <message name> ( <argument type> <argument name> ) {
    <statements to execute when message received>
}
```

- In a method, "this" names the object that received the message.
- "this" and argument name are *local scratch variables*, created when the method is invoked, and thrown away when it finishes.

Invoking Methods, in Detail

To evaluate a message like

```
home . addTo ( aWindow )
```

Java goes through the following steps:

1. Evaluate the receiver object and argument *expressions*, to compute the receiver and argument *objects*
2. Find the method in the class of the receiver object
3. Create a scratch area for running the method
4. In the scratch area:
   1. bind "this" to the receiver *object*
   2. bind the argument name (if any) to the argument *object*
5. Run the method's body
6. Throw away the scratch area and its bindings

Testing Method addTo

- In Jeva, we could just type the following

```
House home = new House( );
GWindow theWindow = new GWindow( );
home.addTo(theWindow);
```

- Where do we put this code in BlueJ?
  - Unfortunately it's hard to create instances of library classes directly in BlueJ ☹
- Solution: Place the test code in the constructor of another class that serves as the "test program"
  - We will replace this later with a standard Java main program, but we want to avoid some of the technicalities for now

Test Code for addTo

```
import uwcse.graphics.*;
/** Test class that, when created, draws a scene in a window */
public class TestScene {
    /** Initialize a new TestScene object,
     * which draws the scene on a new window */
    public TestScene( ) {
      GWindow gw = new GWindow( );
      House abode = new House( );
      abode.addTo(gw);
    }
}
```

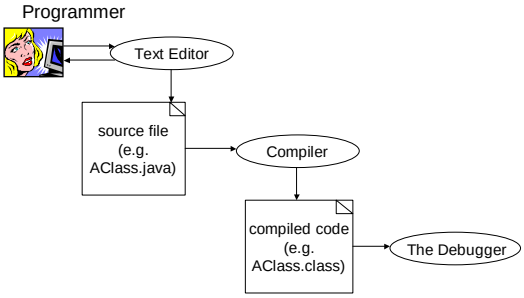
- **Now we can test addTo by creating a TestScene object!**

Diagramming Execution of addTo

The Debugger

- **We can watch a program execute using a *debugger*.**
- **Every debugger supports the following four fundamental concepts:**
  - **We can set a *breakpoint*. Execution pauses when a breakpoint is reached**
  - **At a breakpoint, we can *continue* execution.**
  - **At a breakpoint, we can make our program take a single *step***
    To the next statement (*step over*), or
    Into a method that we're calling (*step into*)
  - **At a breakpoint, we can *inspect* data.**

Tools in Pictures: The Debugger



Debugging in BlueJ

- **Process:**
  - **Double-click on a class to look at its Java source code**
  - **Click in the left white column next to lines where you want execution to stop: *breakpoints***
  - **Then run code (e.g. create objects, send messages)**
- **If a breakpoint is reached, a debugger window appears, showing the current values of local variables and instance variables of the receiver object.**
  - **You can step (over), step into, continue, or terminate**
  - **You can set or remove breakpoints**
  - **You can inspect the objects that the variables refer to**
- **Testing strategy when you first run something: set a breakpoint at the start and watching what it does as you step.**

Documentation

- **Raw code isn't always clear to human readers.**
  - **What is the code intended to accomplish?**
  - **What are the messages I can send to a class in a library?**
- **Need to write comments and other documentation to *describe the interface* and *explain the algorithm* to humans.**
- **Java has two ways of writing comments:**

```
// This is a comment to the end of the line
/* This is a
   comment than
   can go over multiple lines */
```

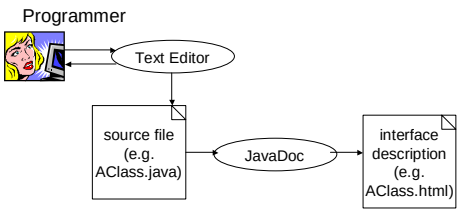
Documenting Constructors and Methods

- The JavaDoc tool looks for comments starting with /** and makes a web page of documentation of the class.

```
/** A class for House shapes. */
public class House {
    /** Create a new House shape. */
    public House( ) { ... }
    /** Display a house on a window.
        @param gw   the window where the house is to be displayed */
    public void addTo(GWindow gw) { ... }
}
```

- Before each class, constructor, and method, explain what it does, and its arguments and results (if any).
  - Should never need to read code to figure out how to use something
- Select "interface" in BlueJ editor window to see the doc pages

Tools In Pictures: JavaDoc



A Constructor With Arguments

- We'd like to be able to create other House objects, but at different parts of the screen.
- Want another constructor, but with *arguments* to say where the house should be located.
- Question:
  - What are the arguments and their types that we want to pass to the constructor?

The New Constructor

```
public class House {
    Rectangle frame;
    Triangle roof;
    public House() { ... }









}
```


Another Message: moveBy

- We can tell a Rectangle to shift its position by sending it the moveBy message.
- We might want to allow users of House objects to tell a House to move, also.
- Questions:
  - How do we make House objects understand a new message?
  - What are the arguments and their types that we want to pass with the message?
  - Do we expect any value to be computed and returned by the message? If so, what is its type?

The moveBy Method

```
public class House {
    Rectangle frame;
    Triangle roof;
    public House() { ... }
    public void addTo(GWindow w) { ... }









}
```

Another Message: getX( )

- We can ask a Rectangle to tell us its x-coordinate by sending it the getX( ) message (getY( ) is similar).
- We might want to allow users of House objects to ask the same question.
- Questions:
  - How do we make House objects understand a new message?
  - What are the arguments and their types that we want to pass with the message?
  - Do we expect any value to be computed and returned by the message? If so, what is its type?

The getX Method

```
public class House {
    Rectangle frame;
    Triangle roof;
    public House() { ... }
    public void addTo(GWindow w) { ... }
    public void moveBy(int deltaX, int deltaY) { ... }









}
```


Return Statements

- To return a computed value back to a caller, use a *return statement*.
- Pattern:

  return <expression> ;
- All methods with non-void result types should end in a return statement! No other methods should!
- Senders can give the returned value a name:

```
House h = new House();
int houseX = h.getX();
```


Protecting an Object's Fields

- Implementation details of an object should not be accessible outside the class (why?)
- To protect an object's fields (parts) from outside access, we can declare that they are private

```
public class House {
    private Rectangle frame;
    private Triangle roof;
    ...
}
```

- This makes outside accesses illegal
  - Good practice: always protect instance variables in the future

Summary

- In the last week we've seen a bunch of colossal ideas
  - Defining new classes
  - Constructors
  - Methods
  - Parameters & results
  - Documentation & comments
- Lots to absorb all at once
- Key concepts behind software development and programming
  - We'll build on this throughout the rest of the course