CSE 142 Summer 2001

How Methods Work

Introduction

- Quick Review
  - Method basics: operational abstractions
  - Methods with parameters
- Today
  - Mental models for understanding methods in action
  - Introduction to scope

Thinking About Methods

- Two perspectives
  - Client view: what the user of the method needs to know to use it
  - Implementer view: what needs to be done to create the method
- The contract between these two views is the *specification* or *interface*
  - Method name, number and types of parameters (if any), result type or void (no explicit result)

```
/** Draw a tree at the given location.
    x the upper left X coordinate of the trunk
    y the upper left Y coordinate of the trunk
*/
public void drawTree(int x, int y)
```


Client View: Substitution

- We can think of a method call as execution of the specification where the values of the arguments are substituted for the parameters.
- call:

```
int offset = 30;
scene.drawTree(2*offset, 140);
```

- means:

```
/** Draw a tree at the given location.
    60 the upper left X coordinate of the trunk
    140 the upper left Y coordinate of the trunk
*/
scene.drawTree(int 60, int 140);
```


Another call of the same method

- Call

```
int location = 95;
scene.drawTree(location, 300);
```

- Meaning

```
/** Draw a tree at the given location.
    95 the upper left X coordinate of the trunk
    300 the upper left Y coordinate of the trunk
*/
scene.drawTree(int 95, int 300);
```

- Idea: A method specification defines a higher-level operation. Particular method calls carry out that operation, with the correct arguments substituted for parameters

Implementer's View: Substitution

- From the implementer's perspective, we can model a method call by replacing the call with the method body, with appropriate substutions
- Specification:

```
/** Draw a tree at the given location.
    x the upper left X coordinate of the trunk
    y the upper left Y coordinate of the trunk */
public void drawTree(int x, int y) {
  int width = 20;
  int height = 100;
  int circleDiameter = 100;
  theWindow.add(new Rectangle(x, y, width, height, Color.blue, true));
  theWindow.add(new Oval(x - (circleDiameter-width)/2,  y - circleDiameter/2,
                         circleDiameter, circleDiameter, Color.green, true));
}
```

Implementer's View: Substitution

- Call

```
scene.drawTree(50, 140);
```

- Meaning

```
int width = 20;
int height = 100;
int circleDiameter = 100;
theWindow.add(new Rectangle(50, 140, width, height, Color.blue, true));
theWindow.add(new Oval(50 - (circleDiameter-width)/2,  140 - circleDiameter/2,
                       circleDiameter, circleDiameter, Color.green, true));
```


Control Flow

- A more operational view is to look at *control flow* – the order in which statements are actually executed

```
int location = 95;                                  // 1
scene.drawTree(location, 300);                      // 7

/**  Draw a tree at the given location …*/
public void drawTree(int x, int y) {
  int width = 20;                                   // 2
  int height = 100;                                 // 3
  int circleDiameter = 100;                         // 4
  theWindow.add(new Rectangle(x, y, width, height, Color.blue, true));          // 5
  theWindow.add(new Oval(x - (circleDiameter-width)/2,  y - circleDiameter/2,
                         circleDiameter, circleDiameter, Color.green, true));  // 6
}
```

- The numbers in the comments show the order in which statements *complete* execution

What Happens

- When we send a message (call a method):
  1) The arguments (they are expressions) are evaluated (left to right).
  2) The method's parameter names are bound to the corresponding values of the arguments.
  3) Control passes to the first statement in the method body.
  4) The body's statements are evaluated until there are no more, or a `return` is encountered.
  5) Control is passed back to the calling statement…
- Key point: Argument evaluation and parameter binding happen ***before*** any statements in the method are executed

Draw the Picture!

```
int location = 95;
scene.drawTree(location, 300);
-------------------------------------
/**  Draw a tree at the given location …*/
public void drawTree(int x, int y) {
  int width = 20;
  int height = 100;
  int circleDiameter = 100;
  theWindow.add(new Rectangle(x, y,
          width, height, Color.blue, true));
  theWindow.add( new Oval(
          x-(circleDiameter-width)/2,
          y - circleDiameter/2,
          circleDiameter, circleDiameter,
          Color.green, true));
}
```


Introduction to Scope

- The *scope* of a name is the region of the program in which that name has a certain meaning (binding).
- Method definitions define a scope: names defined inside of a method (either parameters or local names) are valid *only* within the body of the method.
- The implication is important: it frees programmers from worrying about name collisions.

Scope in Pictures

```
class Scene {
  public void drawHouse( ) {
    int x = 10; int y = 20; …
    theWindow.add(new Rectangle(x, y, …));
    …
  }

  public void drawTree(int x, int y, …) {
    …
    theWindow.add(new Rectangle(x, y, …));
    …
  }

}
```

- Each method defines its own scope: hence no confusion about "x" or "y"

Nesting Scopes

- Scopes may be nested. A class definition defines a scope, with method definition scopes nested inside it.

```
class Scene {
  GWindow theWindow;

  public void drawHouse( ) {
    int x = 10; int y = 20; ...
    theWindow = newGWindow( );
    theWindow.add(new Rectangle(x, y, ...)); ...
  }

  public void drawTree(int x, int y, ...) {
    ...
    theWindow.add(new Rectangle(x, y, ...));  ...
  }
}
```


Which Name?

- Name resolution (figuring out which name to use) proceeds as follows:
  - Starting in the current scope, was the name defined in this scope?
  - If not, go to the enclosing scope and look for the name there.
- This explains how we can use the name theWindow in the previous example even though it was not defined in the scope of drawHouse( ) or drawTree( )
- This applies to both method and object names