CS415

INTRODUCTION TO COMPUTER SCIENCE

SPRING 2017

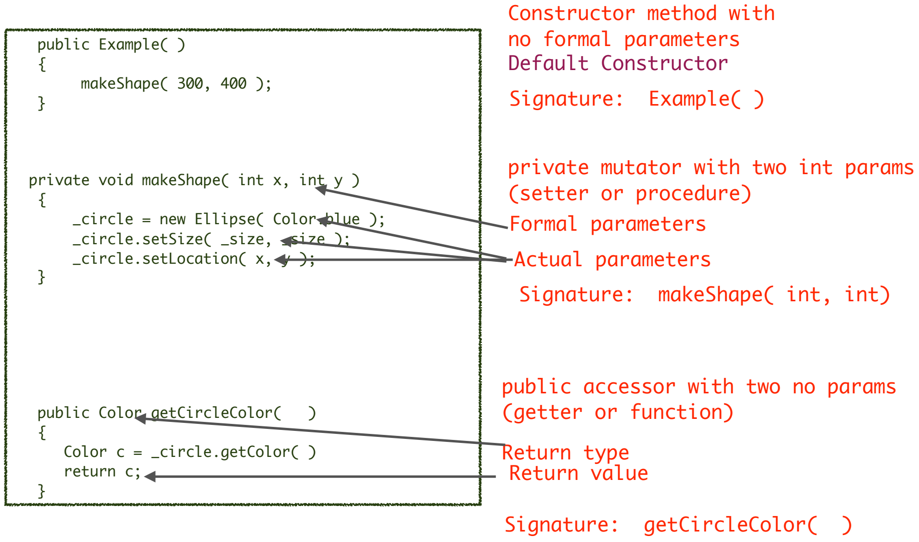
6 CREATING YOUR OWN OBJECTS

SECTION 2.4-2.5

PREVIEW

- Designing and implementing your own objects.
- Trying out your object:
  - DrJava interactions
  - Class "testing" main
  - Java Application
- UML: a way to describe classes and the relationships between them.
- Improving your design.
- Instance variables, local variables, parameters.

METHOD TERMINOLOGY



```
 public Example( )
 {
      makeShape( 300, 400 );
 }


private void makeShape( int x, int y )
 {
     _circle = new Ellipse( Color.blue );
     _circle.setSize( _size, _size );
     _circle.setLocation( x, y );
 }


 public Color getCircleColor(   )
 {
    Color c = _circle.getColor( )
    return c;
 }
```

Constructor method with
no formal parameters
Default Constructor

Signature: Example( )

private mutator with two int params
(setter or procedure)
Formal parameters
Actual parameters

Signature: makeShape(int, int)

public accessor with two no params
(getter or function)
Return type
Return value

Signature: getCircleColor( )

METHOD OVERLOADING

- Methods are <u>not</u> identified by their name but by their signature
- So **makeTarget(Color c)** and **makeTarget(int x, int y)** are considered different methods.
- When we have multiple methods with the same name (but different signatures) we say that the method name has been overloaded.
- Despite the connotation overloading is a good thing.

TYPES OF VARIABLES

```
public class Thing
{
    private Color myColor;

    public Thing( Color aColor )
    {
          Color tempColor = aColor.darker();
          myColor = aColor;

     .......
```

	Example	Scope	Lifetime	Use
Instance Variable	myColor	Class	Instance	Property of Object
Formal Parameter	aColor	method	method	Value passed into method
Local Variable	tempColor	method	method	Temporary value in method

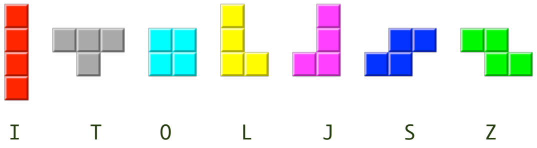
OO MODELS AND JAVA

- An OO model contains objects that interact
- Objects have state and behavior.

Model	Java
Object	We define a Class to represent a type of object
Objects have state	We define an objects state with private instance variables
Objects have behavior	We define an objects behavior with public methods

IMPLEMENTING OBJECTS IN JAVA

- Remember:
  - OOP models require objects.
  - In Java we define Classes in order to create objects
- To see how this is done we will define a class of Tetris "LShapes"



ANALYZING THE OBJECT LSHAPE

- "An LShape should be composed of 4 tiles, it should have a color and a size and It should be able to fall"
- What are the Shape properties ?
  - tiles, color, size, ... these will become instance variables
- What are Shape capabilities?
  - construct itself, fall, getYLocation ,... these will become methods

LShape.java

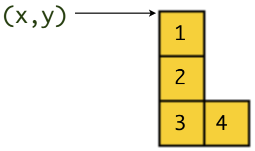
```
import wheels.users.*;
import java.awt.Color;

/**
 * LShape models a Tetris L Shape that can fall.
 * @author cs415
 */
public class LShape
{

     //------------- instance variables --------------------


     //------------- constructors --------------------------


     //------------- other methods -------------------------



} // end of class LShape
```

LSHAPE INSTANCE VARIABLES

```
    //------------- instance variables --------------------

    private Rectangle _tile1, _tile2, _tile3, _tile4;
    //     |1   |
    //     |2   |
    //     |3 4 |

    private Color _myColor;
    private int _tileSize;
```

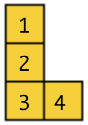
LSHAPE CONSTRUCTOR

```
//------------- constructors --------------------------
public LShape(int x, int y, int size)
{
     _tileSize = size;
     _myColor = Color.orange;  // using “Tetris World” colors
     _tile1 = new Rectangle( x,              y                );
     _tile2 = new Rectangle( x,              y + _tileSize     );
     _tile3 = new Rectangle( x,              y + _tileSize * 2 );
     _tile4 = new Rectangle( x + _tileSize,  y + _tileSize * 2 );

     _tile1.setFillColor( _myColor );
     _tile1.setFrameColor( Color.black );
     _tile2.setFillColor( _myColor );
     _tile2.setFrameColor( Color.black );
     _tile3.setFillColor( _myColor );
     _tile3.setFrameColor( Color.black );
     _tile4.setFillColor( _myColor );
     _tile4.setFrameColor( Color.black );

     _tile1.setSize( _tileSize, _tileSize );
     _tile2.setSize( _tileSize, _tileSize );
     _tile3.setSize( _tileSize, _tileSize );
     _tile4.setSize( _tileSize, _tileSize );
}
```



```
LShape s = new LShape( 20, 50, 25 );
```

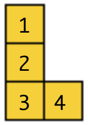
fall( )



```
/**
* Make the LShape “fall” one tileSize.
*
*
*/
public void fall( )
{
    _tile1.setLocation( _tile1.getXLocation(), _tile1.getYLocation() + _tileSize );
    _tile2.setLocation( _tile2.getXLocation(), _tile2.getYLocation() + _tileSize );
    _tile3.setLocation( _tile3.getXLocation(), _tile3.getYLocation() + _tileSize );
    _tile4.setLocation( _tile4.getXLocation(), _tile4.getYLocation() + _tileSize );
}
```

```
s.fall( );
```

getYLocation( )



```
/**
* Return the LShape y location.
*
*
*/
public int getYLocation( )
{
      // the location of the Shape is the location of _tile1.
      return _tile1.getYlocation();
}
```

```
int y = s.getYLocation();
```

TRYING OUT LSHAPE

- We have a few ways to try out Lshape
  - Use the DrJava interactions pane: a quick and easy way to test
    - Create a few Tiles and send them messages
  - Write a main method in the LShape class
    - Write a main method in the Tile Class: a "testing main"
  - Write a Java application that uses the LShape: a tetris game?
    - This is the real reason for creating LShape

LSHAPE IN DRJAVA

- In the DrJava Interactions pane enter the following

```
Welcome to DrJava
>import wheelsunh.users.*;
>Frame display = new Frame();
>LShape one = new LShape(200,200);
>LShape two = new LShape(300,200);
>one.fall();
>one.fall();
```

A MAIN METHOD IN

- Add a main method to the LShape class

```
.
.
.
public static void main( String[] args )
{
   Frame f = new Frame();
   LShape one = new LShape( 200, 300 );
   LShape two = new LShape( 200, 200 );
   one.fall();
}
.
.
.
```

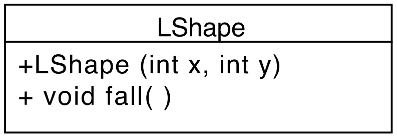
AN LSHAPE APPLICATION

- Write an application that uses LShape

```
/**
 * This class tries out the LShape object.
 */
public class LShapeTester
{
     private LShape one,two;

     LShapeTester()
     {
         Lshape one = new LShape( 200, 200 );
         LShape two = new LShape( 300, 200 );
         one.fall();
     }
     public static void main( String[] args )
     {
         new Frame();
          LShapeTester app = new LShapeTester();
     }

} // end of class LShapeTester
```

UML

- We can describe the classes in our model and the relationships among them with UML diagrams.
- We describe the LShape class as follows:

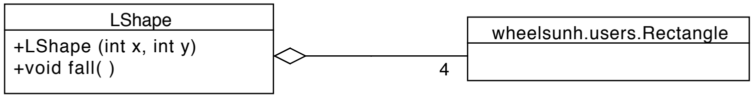


LShape

+LShape (int x, int y)
+ void fall( )

UML

- UML can be used to show the relationships among the objects.
- LShape "has" four tiles; these are its components.
- The LShape is composed of its tiles, it is the container of the component Rectangles.
- We diagram the component/container relationship as follows.



LShape

+LShape (int x, int y)
+void fall( )

wheelsunh.users.Rectangle

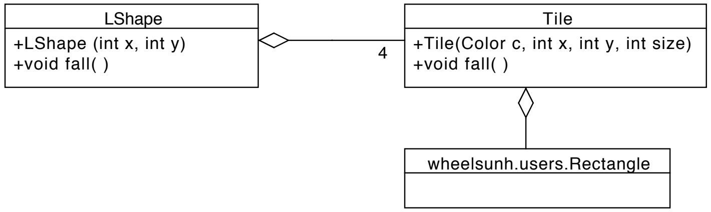
CHANGING THE DESIGN

- The code for LShape is fairly complicated.
- To create each tile we create its rectangle, frame it, color it, size it and position it.
- To make each tile fall get its position and calculate its new position.
- If we had a Tile object that could do these things for itself we could simplify LShape. $\sum$

Dream... If we had a Tile object with the proper behavior we could simplify LShape

```
// instance variables: Tiles instead of Rectangles
private Tile _tile1, _tile2, _tile3, _tile4;
.
.
.
public LShape( int x, int y )
{
    tileSize = 20;
    myColor = Color.orange;
    _tile1 = new Tile( _myColor, x,            y,                 _tileSize );
    _tile2 = new Tile( _myColor, x,            y + _tileSize,     _tileSize );
    _tile3 = new Tile( _myColor, x,            y + _tileSize * 2, _tileSize );
    _tile4 = new Tile( _myColor, x + tileSize, y + _tileSize * 2, _tileSize );
}

public void fall()
{
    _tile1.fall();
    _tile2.fall();
    _tile3.fall();
    _tile4.fall();
}
```

THE NEW DESIGN


LShape
+LShape (int x, int y)
+void fall( )
4
Tile
+Tile(Color c, int x, int y, int size)
+void fall( )
wheelsunh.users.Rectangle

TILE.JAVA

```
import wheelsunh.users.*;
import java.awt.Color;

/**
 * Tile models a Tetris tile with a given color, location and size.
 * The tile can fall.
 * @author cs415
 */
public class Tile
{
     private Rectangle _body;
     private Color _color;
     private int _size;
     private int _x, _y;

     /**
     * Constructor
     */
    public Tile( Color color, int x, int y, int size )
     {
           // to be done
     }

    /**
    * Fall.
    */
    public void fall()
     {
           // to be done
     }
}
```

TILE METHODS

```
public Tile( java.awt.Color color, int x, int y, int size )
{
   _color = color;
   _x = x;
   _y = y;
   _size = size;
   _size = size;
   _body = new Rectangle( _x, _y );
   _body.setColor( _color );
   _body.setSize( _size, _size );
   _body.setFrameColor(  Color.black );
}

public void fall()
{
   _body.setLocation( _body.getXLocation(), _body.getYLocation() + _size );
}
```

REVIEW

- Designing and implementing your own objects.
- Trying out your object:
  - DrJava interactions
  - Class "testing" main
  - Java Application
- UML: a way to describe classes and the relationships between them.
- Improving your design.
- Instance variables, local variables, parameters.

NEXT TIME

- Some systems may have many objects, how can we manage this complexity?
- In everyday life we organize the objects around us by grouping them together based on shared features.
  - Each object then shares most of its properties and capabilities with its group.
  - Each object specializes the group by adding only a few specialized features
- This is an ability we want in OOP.
- Read Chapter 3.