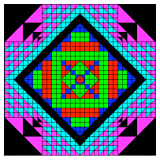
PictureWorld

A world of fruitful methods

Tuesday, September 19, 2006



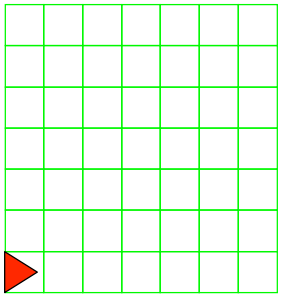
CS111 Computer Programming

Department of Computer Science
Wellesley College

Buggles and BuggleWorld

Primitives

Simple values: e.g, integers, booleans, …
Black-box objects: e.g., buggles, points, colors
Black-box methods: forward(), left(), …



Means of Combination

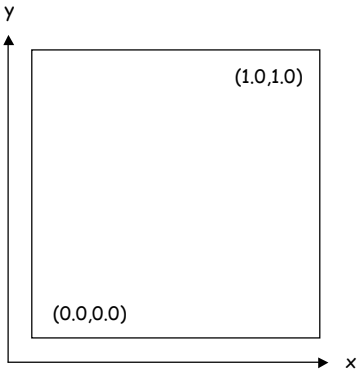
Arithmetic on numbers: 1 + 2
Instance variable reference: p.x
Class variable reference: Color.red
Void method invocation: becky.forward()
Nonvoid method invocation: becky.getColor()
Constructor method invocation: new Point(1,2)
Sequences of statements:
    bob.forward(3); bob.left();

Means of abstraction

Creating our own methods: rollover(), dogHouse()
Creating our own classes: LetterBuggle

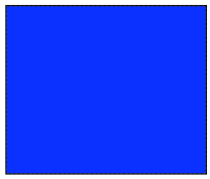
PictureWorld

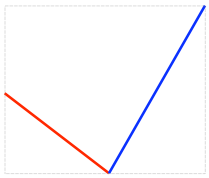
- A world for studying (1) fruitful methods and (2) notion that complex structures can be built from simple primitives using powerful means of combination and means of abstraction.
- Each picture exists within the unit square, and understands one method: how to stretch itself to fill a given rectangle in the applet.
- PictureWorld's GUI is much simplier than BuggleWorld's.



Some "Primitive" Pictures

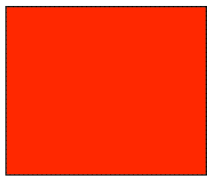
bp
(blue patch)

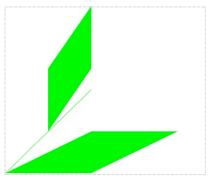




mark

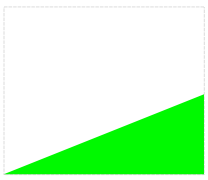
rp
(red patch)





leaves

gw
(green
wedge)



empty

Adding Pictures to a PictureWorld

```
public class SimplePictureWorld extends PictureWorld {

  public void initializePictureChoices() {

    // Create new "primitive" pictures
    // (they're not really primitive, as we'll see later)
    Picture bp = this.patch(Color.blue); // blue patch
    Picture rp = this.patch(Color.red); // red patch
    Picture mark = this.checkmark(Color.red, Color.blue); // red/blue check
    Picture gw = this.wedge(Color.green); // green wedge
    Picture leaves = this.twoLeaves(Color.green); // green leaves

    // Add pictures to the menu
    this.addPictureChoice("blue patch", bp);
    this.addPictureChoice("red patch", rp);
    this.addPictureChoice("mark", mark);
    this.addPictureChoice("green wedge", gw);
    this.addPictureChoice("leaves", leaves);

    // Notes:
    // 1. "this." is optional; can omit it!
    // 2. Can always use Picture expression in place of name.
    // For example:
    //   addPictureChoice("mark", checkMark(Color.red, Color.blue))
  }
  ...
}
```

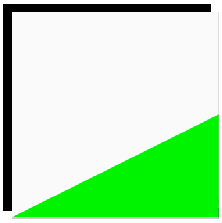

PictureWorld contract

```
// Automatically invoked when a PictureWorld applet is created
public void initializePictureChoices();

// Adds name to the menu and associates it with pic
public void addPictureChoice(String name, Picture pic);

public Picture empty();  // Returns an empty picture
```

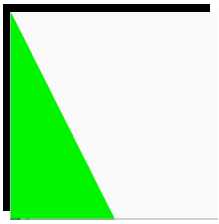
We'll now explore many more methods in the contract …

Rotating Pictures

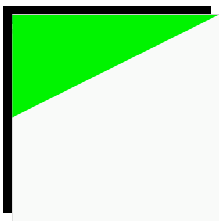
```
public Picture clockwise90(Picture p)  // Returns p rotated 90° clockwise
public Picture clockwise180(Picture p) // Returns p rotated 180° clockwise
public Picture clockwise270(Picture p) // Returns p rotated 270° clockwise
```



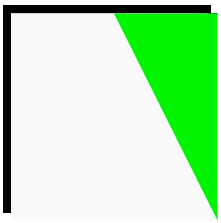
gw



clockwise90(gw)



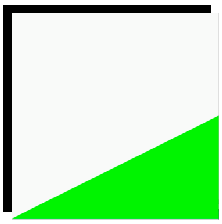
clockwise180(gw)



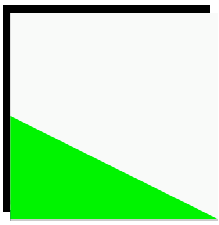
clockwise270(gw)

Flipping Pictures

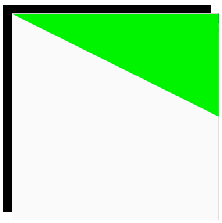
```
public Picture flipHorizontally(Picture p) // Returns p flipped across horiz axis
public Picture flipVertically(Picture p) // Returns p flipped across vert axis
public Picture flipDiagonally(Picture p) // Returns p flipped acros diag axis
```



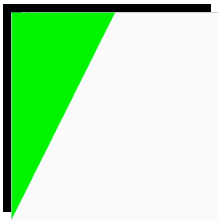
gw



flipVertically(gw)



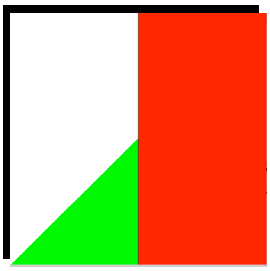
flipHorizontally(gw)



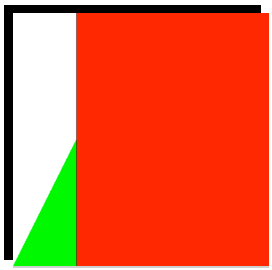
flipDiagonally(gw)

Putting one picture beside another

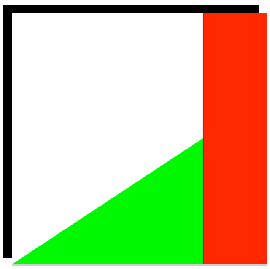
```
// Returns picture resulting from putting p1 beside p2
public Picture beside(Picture p1, Picture p2)

// Returns picture resulting from putting p1 beside p2,
// where p1 takes give fraction of rectangle.
public Picture beside(Picture p1, Picture p2, double fraction)
```



beside(gw,rp)



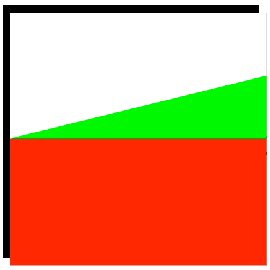
beside(gw,rp,0.25)



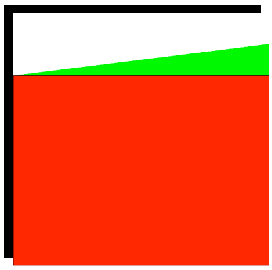
beside(gw,rp,0.75)

Putting one picture above another

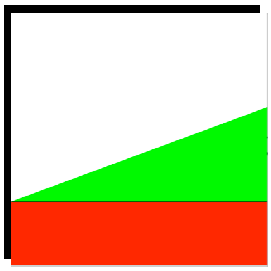
```
// Returns picture resulting from putting p1 above p2
public Picture above(Picture p1, Picture p2)

// Returns picture resulting from putting p1 above p2,
// where p1 takes give fraction of rectangle.
public Picture above(Picture p1, Picture p2, double fraction)
```



above(gw,rp)



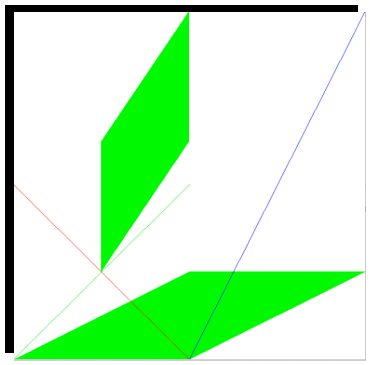
above(gw,rp,0.25)



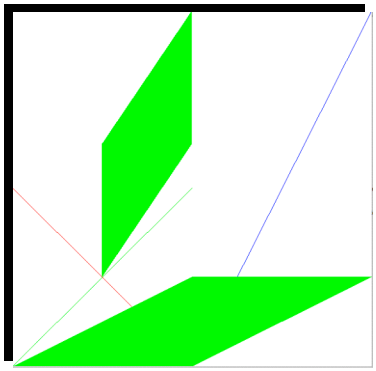
above(gw,rp,0.75)

Putting one picture over another

```
// Returns picture resulting from overlaying p1 on top of p2
public Picture overlay(Picture p1, Picture p2)
```



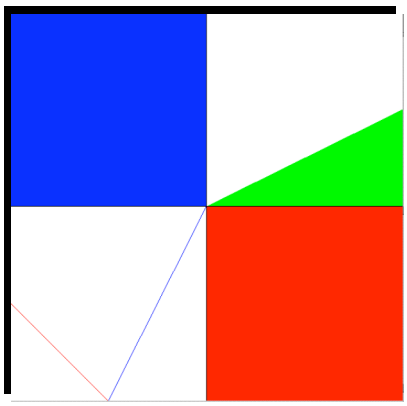
overlay(mark,leaves)



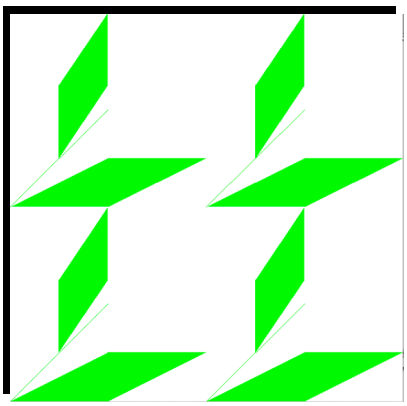
overlay(leaves,mark)

Combining Four Pictures

```
public Picture fourPics(Picture p1, Picture p2, Picture p3, Picture p4) {
    return above(beside(p1, p2), beside(p3,p4));
}

public Picture fourSame(Picture p) {
    return fourPics(p,p,p,p);
}
```



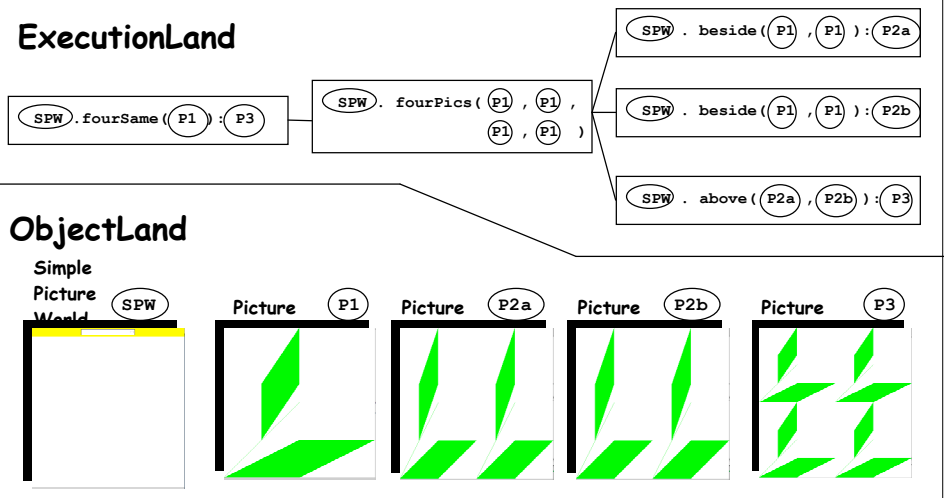
fourPics(bp,gw,mark,rp)



fourSame(leaves)

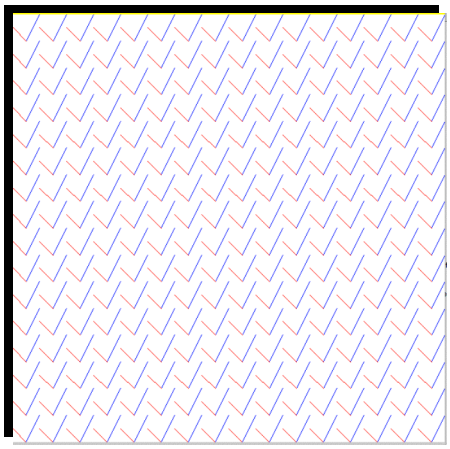
Invocation Trees

An invocation tree is an abbreviation of a tree of JEM frames. It is especially helpful for understanding computations involving fruitful methods.

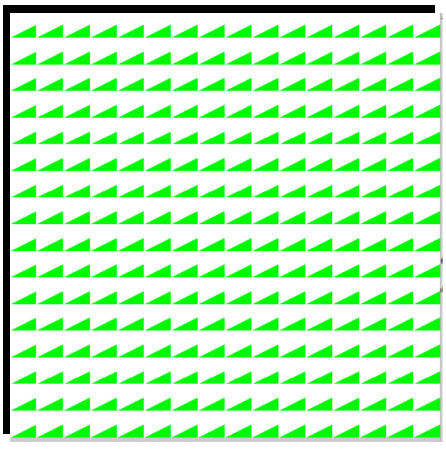


Repeated Tiling

```
public Picture tiling (Picture p) {
    return fourSame(fourSame(fourSame(fourSame(p))));
}
```



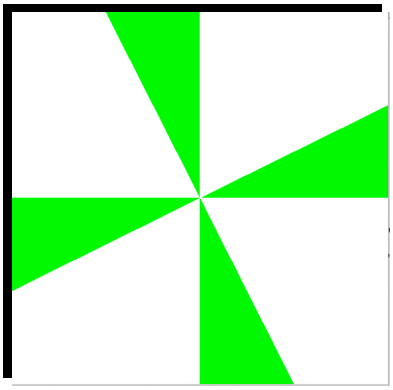
tiling(mark)



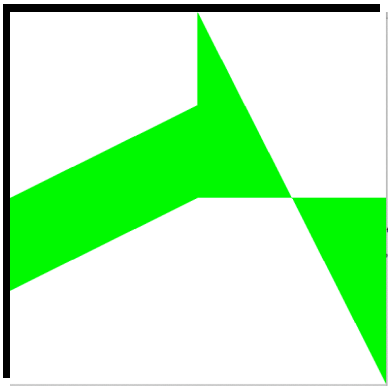
tiling(gw)

Rotation Combinators

```
public Picture rotations (Picture p) {
    return fourPics(clockwise270(p), p, clockwise180(p), clockwise90(p));
}

public Picture rotations2 (Picture p) {
    return fourPics(p, clockwise90(p), clockwise180(p), clockwise270(p));
}
```



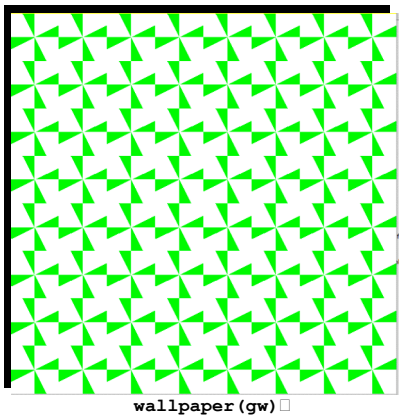
rotations(gw)



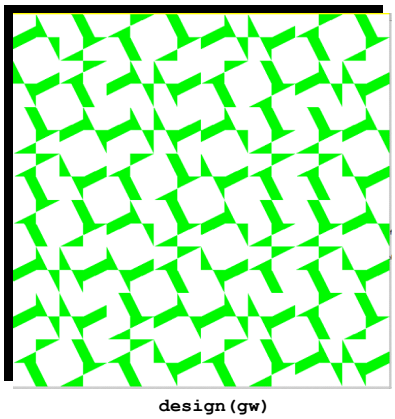
rotations2(gw)

A Simple Recipe for Complexity

```
public Picture wallpaper (Picture p) {
    return rotations(rotations(rotations(rotations(p))));
}

public Picture design (Picture p) {
    return rotations2(rotations2(rotations2(rotations2(p))));
}
```



wallpaper(gw)



design(gw)