Following The Green Goblin

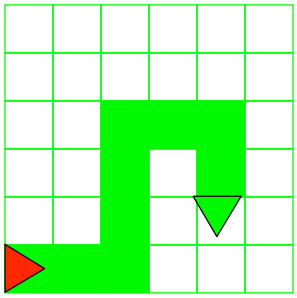
Booleans and their friends

Friday, September 29, 2006

CS111 Computer Programming

Department of Computer Science
Wellesley College

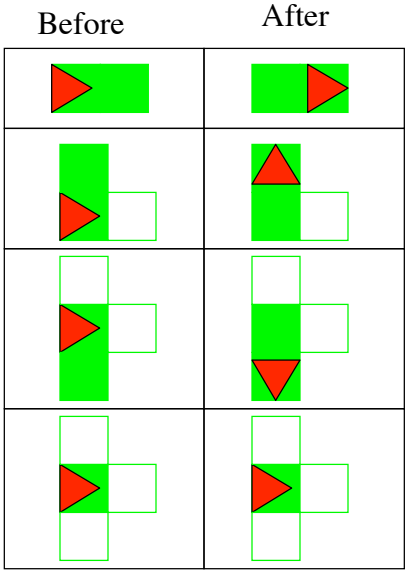
On the trail of the Green Goblin

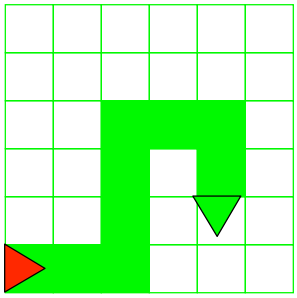


(Assume goblin never doubles back or crosses its own path.)

Possible Situations

(relative to current Buggle orientation)





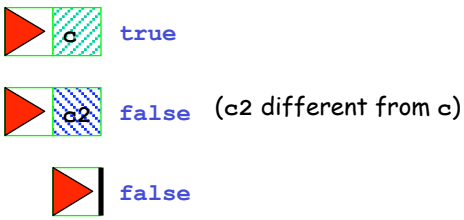
Wishful Thinking

Suppose we have a method with the following specification:

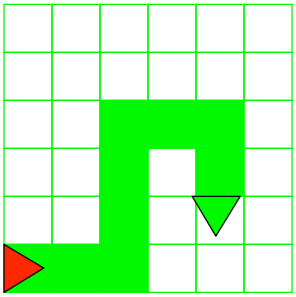
```
public boolean isFacingTrail (Color c);
```

Returns true if this buggle is facing a cell filled with color c; otherwise returns false. Leaves the state of the buggle unchanged.

Pictorial case analysis:



followGoblin()

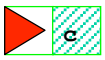


```
public void followGoblin()
{
    if (isFacingTrail(Color.green)) {
        forward();
    } else {
        left();
        if (isFacingTrail(Color.green)) {
            forward();
        } else {
            right(); right();
            if (isFacingTrail(Color.green)) {
                forward();
            } else {
                left(); // I must be right being him!
            }
        }
    }
}
```


isFacingTrail(Color c)

```
public boolean isFacingTrail(Color c)
{
  if (isFacingWall()) { // no need for "== true"
    return false;
   } else { // cell in front of me
    brushUp();
    forward();
    if (getCellColor().equals(c)) {
      backward(); // undo forward();
      brushDown(); // undo brushUp();
      return true;
    } else {
      backward(); // undo forward();
      brushDown();// undo brushUp();
      return false;
    }
  }
}
```

false



true

false

(c2 different from c)

Boolean Expressions & Declarations

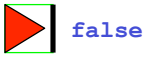
- Like int, float, double, and char, boolean is a Java primitive data type. A boolean can have only one of two values; true or false.
- We have already seen many examples of boolean expressions:

```
true  becky.isOverBagel()  spidey.isFacingTrial(Color.green)
```

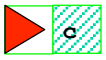
- We can declare boolean variables like any other variables:

```
boolean result = getCellColor().equals(c);

boolean atWall;
atWall = greenGoblin.isFacingWall();
```


A More Compact isFacingTrail(Color c)




```
public boolean isFacingTrail(Color c)
{
    if (isFacingWall()) {
        return false;
    } else { // cell in front of me
        brushUp();
        forward();
        boolean result = getCellColor().equals(c);
        backward(); // undo forward();
        brushDown(); // undo backward();
        return result;
    }
}
```



(c2 different from c)

Testing for equality in Java

- Primitive types (int, double, float, char, boolean) use the == operator*. For example:

1 == 1	true
1 == 2	false
false == false	true
true == false	false

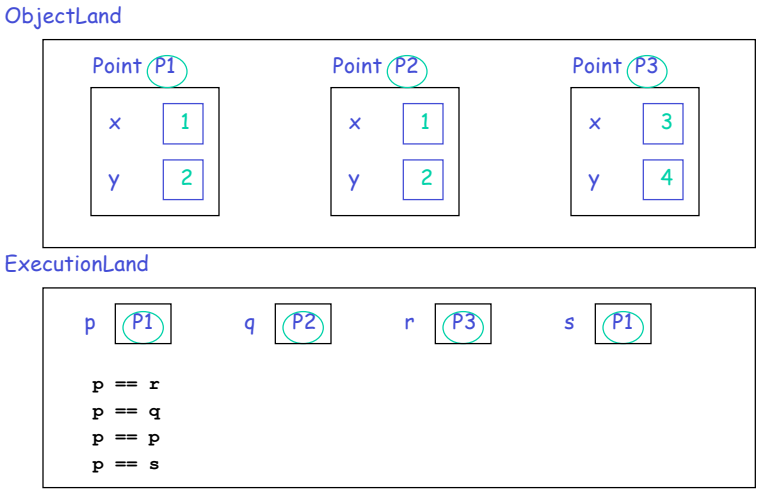
- Object equality can be tested in two ways:

1) == tests object identity: do two object references refer to the same object in ObjectLand?

2) .equals() typically tests structural equality of objects: do two objects have the same state?

*== has number of friends for numerical types: < (less than), > (greater than) , <= (less than or equal), >= (greater than or equal), != (not equal).

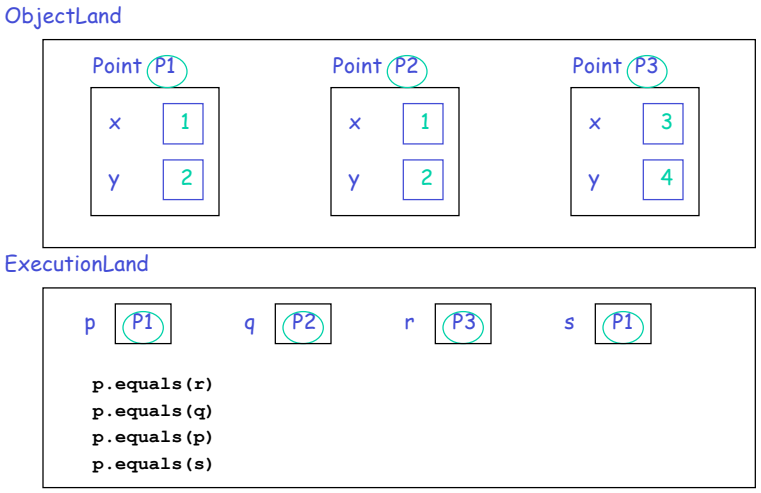
Testing Object Identity via ==



.equals() Tests Structural Equality

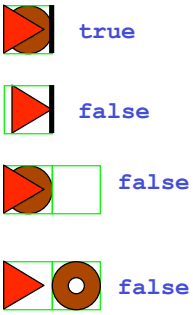
```
class Point // Represents an (x,y) point
{ // instance variables:
  public int x;
  public int y;

  // constructor method:
  public Point(int x_coord, int y_coord)
  {
    this.x = x_coord;
    this.y = y_coord;
  }

  // equals() instance method
  // (usually takes an Object parameter; for simplicity,
  //  this version is specialized to a Point param)
  public boolean equals (Point p)
  {
    if (this.x == p.x) {
      if (this.y == p.y) {
        return true;
      } else {
        return false;
      }
    } else {
      return false;
    }
  }
}
```


Testing Object Equality via .equals()



The Conjunction (AND) Operator: && †

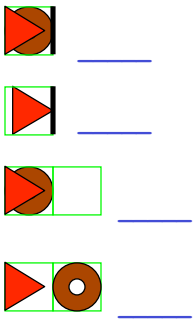
- The expression exp1 && exp2 is true precisely when the expression exp1 is true AND the expression exp2 is true.
- For example, `isFacingWall() && isOverBagel()`



†A single & means something else

The Disjunction (OR) Operator: || †

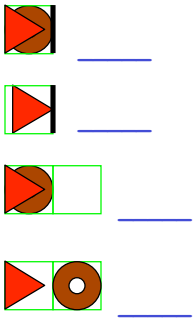
- The expression exp1 || exp2 is true precisely when either expression exp1 is true OR expression exp2 is true OR both.
- For example, `isFacingWall() || isOverBagel()`



†A single | means something else

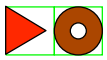
The Negation (NOT) Operator: !

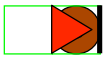
- The expression ! exp is true precisely when the expression exp is false.
- For example, `!isFacingWall()`

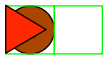


These operators can be combined

`!isFacingTrail(Color.green)`
`&& !isOverBagel()`







`isFacingTrail(Color.green)`
`|| isFacingWall()`

Boolean expressions can be very expressive

```
boolean honor = ((class == 2006) && (gpa >= 3.8))
                || (class == 2007) && (gpa >= 3.75));


if ((betty.getPosition().x == 1)
    && (betty.getPosition().y == 2)) {
  betty.setColor(Color.green);
} else {
  betty.setColor(Color.blue);
}


return (becky.getColor().equals(Color.red))
       || (becky.getColor().equals(Color.blue))
       || (becky.getColor().equals(Color.yellow));


bob.setBrush((! (bernice.getHeading().equals(Direction.EAST)))
             && ((becky.getHeading().equals(Direction.WEST))
                 || (betty.getHeading().equals(Direction.SOUTH))));
```


&& and || are Short-Circuit Operators

Second (right-hand) operand expression is not evaluated if result is determined by first (left-hand) operand expression.

Examples: Suppose `x` is 7 and `y` is 0:

```
(x > 5) || ((x/y) > 2)   true
(x < 5) || ((x/y) > 2)   division-by-zero error
(x < 5) && ((x/y) > 2)   false
(x > 5) && ((x/y) > 2)   division-by-zero error
```

Standard Simplifications

Suppose BE ranges over boolean expressions and S ranges over statements. You should make the following simplifications in your code:

BE == true	BE
BE == false	! BE
if (BE) {return true;} else {return false;}	return BE;
if (BE) {return false;} else {return true;}	return ! BE;
if (BE1) {return BE2;} else {return false;}	return BE1 && BE2;
if (BE1) {return true;} else {return BE2;}	return BE1 || BE2;
boolean result = BE; return result;	return BE
if (BE) {S; return true;} else {S; return false;}	boolean result = BE; S; return result;

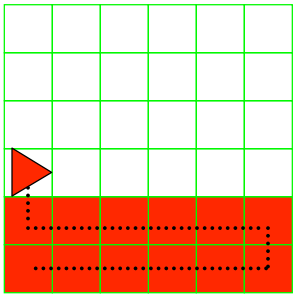
Example: Simplifying .equals() for Points

```
public boolean equals (Point p)
{
  if (this.x == p.x) {
    if (this.y == p.y) {
      return true;
    } else {
      return false;
    }
  } else {
    return false;
  }
}
```

```
public boolean equals (Point p)
{
  if (this.x == p.x) {
    return (this.y == p.y)
  } else {
    return false;
  }
}
```

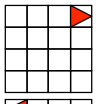
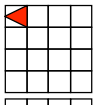
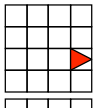
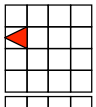
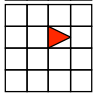
```
public boolean equals (Point p)
{
  return (this.x == p.x)
         && (this.y == p.y);
}
```

SnakeWorld

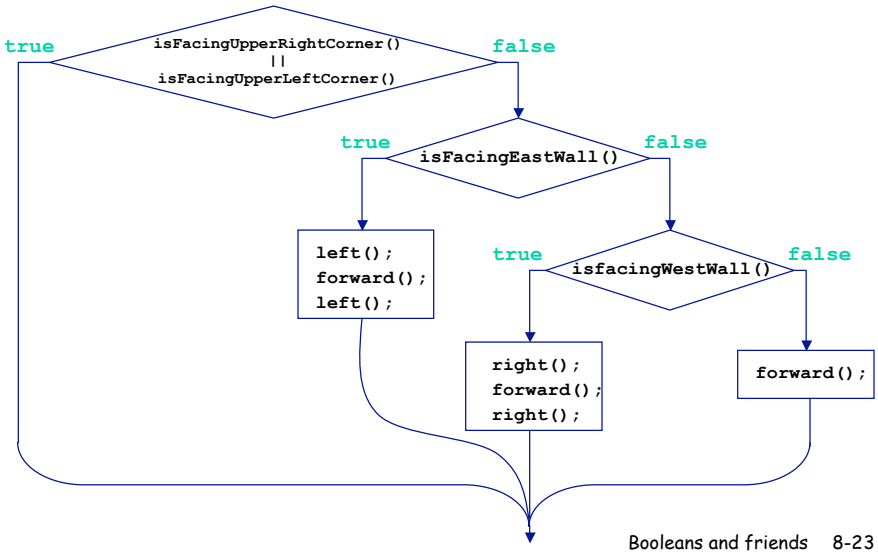


*We help Sandra buggle to traverse every cell in BuggleWorld in a boustrophedonic fashion.

The possibilities

	State	Action
top right corner, facing EAST		nothing
top left corner, facing WEST		nothing
at wall, facing EAST		
at wall, facing WEST		
no wall		move forward

Control diagram



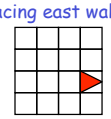
In Java

```
public class SnakeBuggle extends TickBuggle
{
  public void tick()
  {
    if (isFacingUpperRightCorner() || isFacingUpperLeftCorner()) {
      // do nothing
    } else if (isFacingEastWall()) {
      left();
      forward();
      left();
    } else if (isFacingWestWall()) {
      right();
      forward();
      right();
    } else { // not facing any wall
      forward();
    }
  }
  // Need to add auxiliary methods
  ...
}
```

Auxiliary methods

```
public boolean isFacingEastWall()
{
  return isfacingWall()
         && (getHeading().equals(Direction.EAST));
}
// isFacingWestWall() is similar

public boolean isFacingUpperRightCorner()
{
  return isfacingEastWall() && isWallToLeft();
}
// isFacingUpperLeftCorner() is similar

public boolean isWallToLeft()
{
  left();
  boolean result = isFacingWall();
  right();
  return result;
}
// isWallToRight() is similar
```

facing east wall



facing upper right corner

A More Efficient, But Less Clear, Solution

```
public void tick()    // Invariant: Buggle starts and finishes
{                     // facing EAST or WEST
    if (isFacingWall()) {
        if (getHeading().equals(Direction.EAST)) {
            left();
            if (isFacingWall()) { // top right corner
                right();          // reset direction to EAST
            } else {              // EAST, at wall
                forward();
                left();
            }
        } else { // must be facing WEST
            right();
            if (isFacingWall()) { // top left corner
               left();            // reset direction to WEST
            } else {              // WEST, at wall
               forward();
               right();
            }
        }
    } else { // no wall
        forward();
    }
}
```

SnakeBuggle

```
public class SnakeWorld extends BuggleWorld
{
  public void run()
  {
    SnakeBuggle sandra = new SnakeBuggle();
    sandra.tick128();
  }
}

public class SnakeBuggle extends TickBuggle
{
  public void tick()
  {
    // snaky code goes here
  }
}
```