Four Big ideas

Programming ways

Fri. Sep. 8, 2006

CS111 Computer Programming

Department of Computer Science
Wellesley College

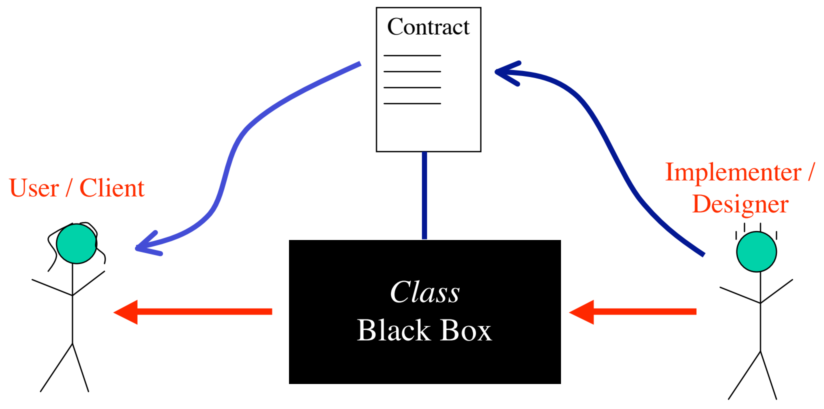
What is Computer Science?

- It's not really about computers .
- It's not really a science.
- It's about imperative ("how to") knowledge as opposed to declarative ("what is") knowledge.
- Imperative knowledge is expressed via algorithms = computational recipes.
- "A computer language … is a novel formal medium for expressing ideas about methodology, not just a way to get a computer to perform operations. Programs are written for people to read, and only incidentally for machines to execute"
  -- *Harold Abelson and Gerald J. Sussman*

Four big ideas

- Four important concepts are at the core of this course:
  1. Abstraction;
  2. Modularity;
  3. Divide, Conquer and Glue;
  4. Models
- These ideas are important in almost every discipline, but they're at the core of CS.
- We will illustrate these ideas in several ways, including Buggles.
- Our goal is to rewire your brain to think in a new way.

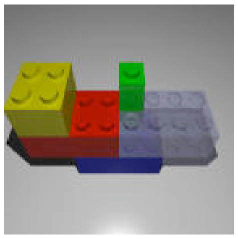
Big idea number 1: Abstraction



*Visit http://cs.wellesley.edu/~cs111/contracts for some useful Java contracts, which are known as Application Programming Interfaces (APIs).

Big idea number 2: Modularity

- Large systems are built from components called modules.
- The interfaces between modules are designed so they can be put together in a mix-and-match way.
- In Java, goal is to design classes for maximum reusability.



Big idea number 3: Divide, conquer & glue

Divide

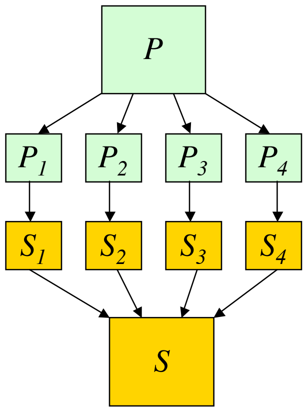
problem P into subproblems.

Conquer

each of the subproblems, &

Glue (combine)

the solutions to the subproblems into a solution S for P.



$$\begin{tikzcd}
 & & P \arrow[dll] \arrow[dl] \arrow[dr] \arrow[drr] & & \\
P_1 \arrow[d] & P_2 \arrow[d] & & P_3 \arrow[d] & P_4 \arrow[d] \\
S_1 \arrow[drr] & S_2 \arrow[dr] & & S_3 \arrow[dl] & S_4 \arrow[dll] \\
 & & S & &
\end{tikzcd}$$

Big idea number 4: Models

- Need simple models to understand complex artifacts and behaviors.
- Throughout this course, we will use a Java Execution Model (JEM) to explain what happens when Java code is executed.
- Motivational example (we'll understand this by the end of class):

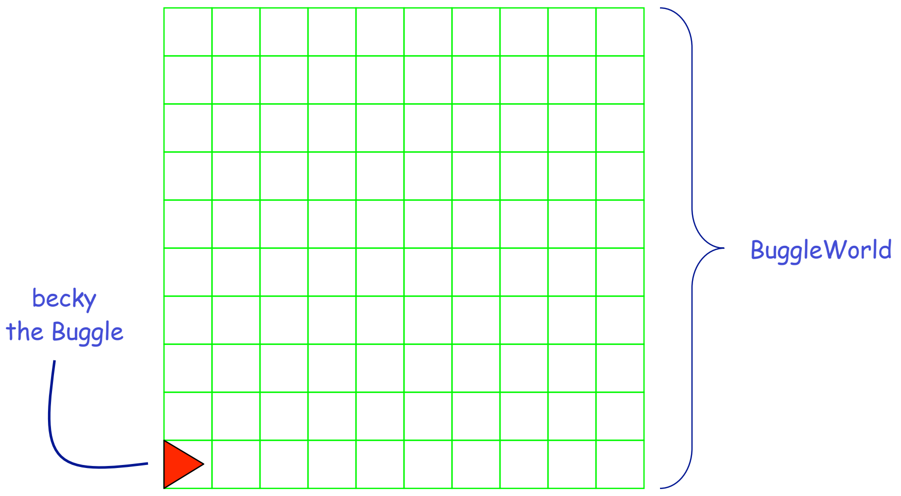
```
Point p1 = new Point(1,2);
Point p2 = new Point(1,2);
Point p3 = p2;
p1.x = p2.y;
p2.y = p3.y;
// What are the coordinates of p1, p2, and p3 now?
```

- Today we introduce one JEM aspect: ObjectLand, the place where Java objects "live".

Object-oriented Terminology

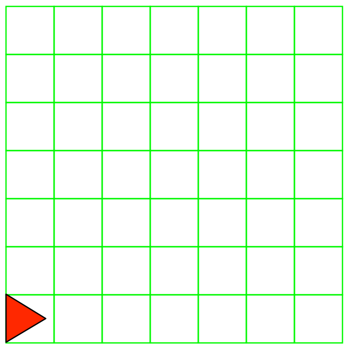
- **Object-oriented** means we create and manipulate program **objects**. Often represent things in the world (my car, you, becky the buggle).
- **Objects** are things that can respond to **messages**. When an object receives a message, it executes the corresponding **method** --- a named sequence of instructions that describes some behavior of an object.
- A **class** is a description of the shared characteristics of a group of objects. It includes the properties (**instance variables**) and methods the objects understand. E.g., a buggle's color or `forward()`.
- An object created based on the class description is an **instance** of the class.
- An object is **mutable** if the state of some of its properties can change over time (e.g., Buggles, Points).
- An object is **immutable** if none of its properties can ever change (e.g. Colors, Directions).

becky in BuggleWorld



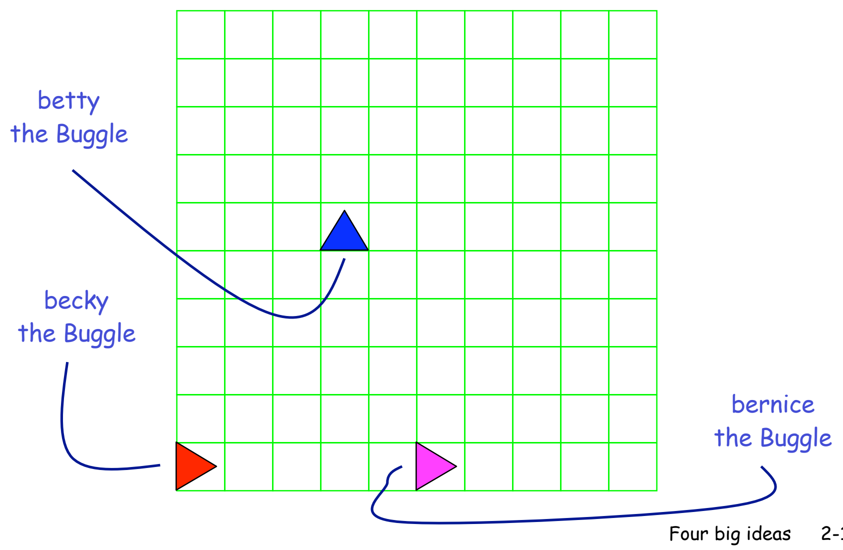
Four properties of Buggles

- position: Where becky sits, specified by an (x, y) coordinate.
- heading: The compass direction becky is facing.
- color: becky and her paint brush's color.
- brushDown: Is becky ready to paint?



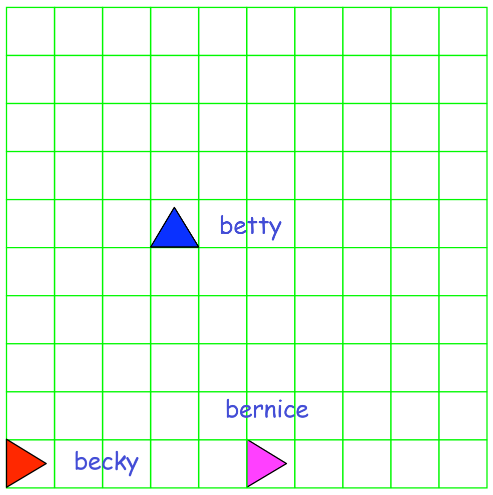
*Collectively these four properties define the state of a Buggle.

becky has company



A class of Buggles

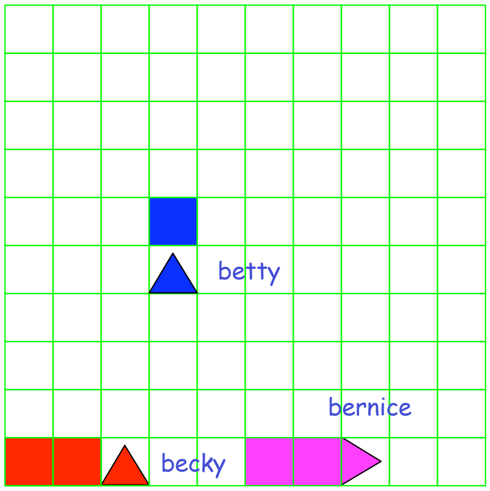
- A class is a collection of objects that have a common "shape" and respond the same way to a known set of messages.
- An object is an instance of a class.



Changing state

We change an object's state by sending it messages.

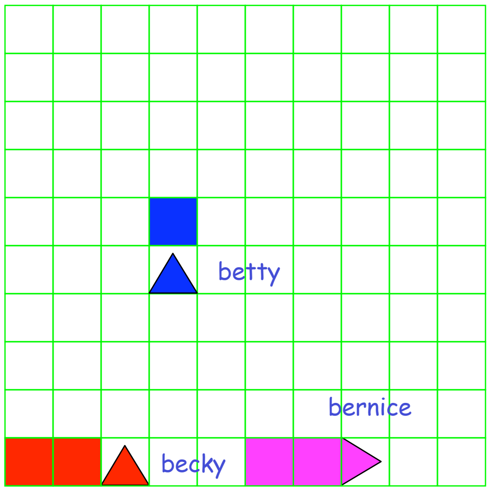
```
becky.forward();
becky.forward();
becky.left();

betty.backward();

bernice.brushDown();
bernice.forward();
bernice.forward();
```



A class is described by

instance variables
  that describe the properties of each class instance; and

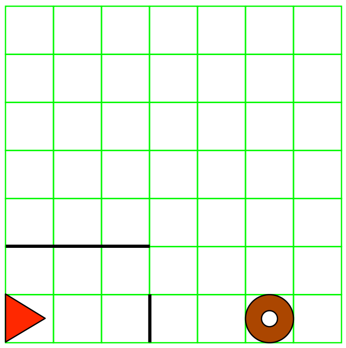
instance methods
  that are the messages to which an instance of the class can respond.



Becky buys a bagel

```
public class BreakfastWorld extends BuggleWorld
{
  public void run ()
  {
      Buggle becky = new Buggle();
      // becky goes outside
      becky.forward(2);
      becky.left();
      becky.forward();
      becky.right();
      becky.forward();
      becky.right();
      becky.forward();
      becky.left();
      // walks to the bagel
      becky.forward(2);
      // and chows down
      becky.pickUpBagel();
  }
}
```

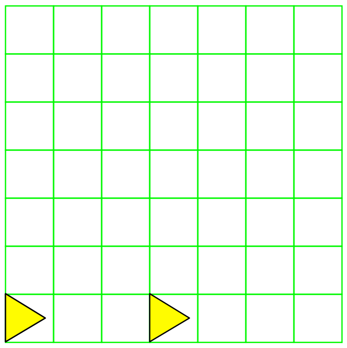
instance messages from Buggle contract



Methods with arguments

- Some methods require additional information, passed as arguments, when invoked.

```
becky.setColor(Color.yellow);


becky.forward(3);
```



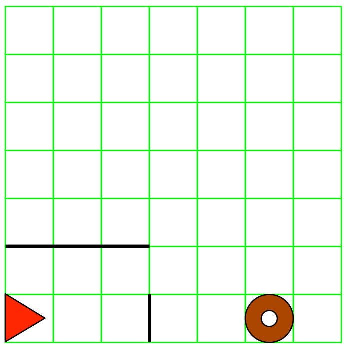
The anatomy of a program

```
public class BreakfastWorld extends BuggleWorld
{
  public void run ()
  {
      Buggle becky = new Buggle();
      // becky goes outside
      becky.forward(2);
      becky.left();
      becky.forward();
      becky.right();
      becky.forward();
      becky.right();
      becky.forward();
      becky.left();
      // walks to the bagel
      becky.forward(2);
      // and chows down
      becky.pickUpBagel();
  } // run()
} // class BreakfastWorld
```

run method

constructing a new Buggle object

comments

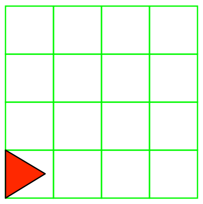


Four big ideas 2-17

A Buggle is born

assignment statement

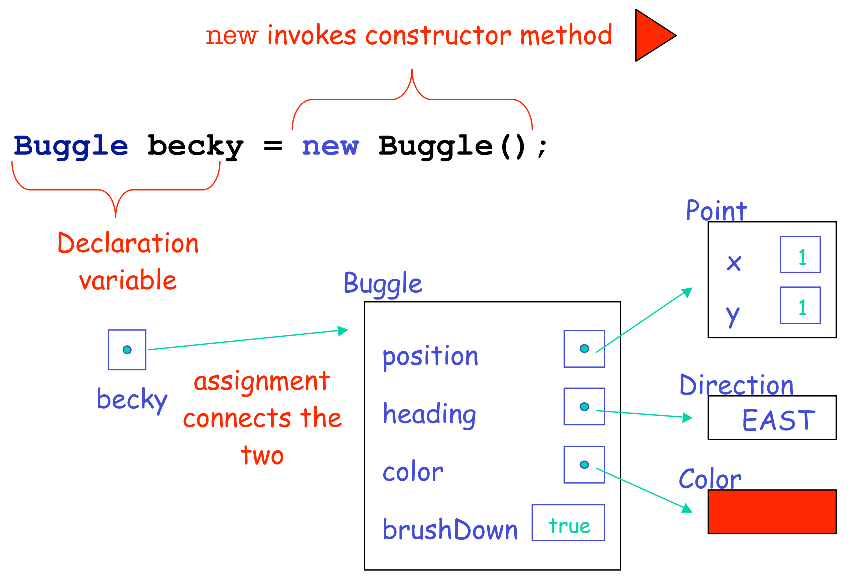
$$\overbrace{\underbrace{\texttt{Buggle becky}}_{\text{variable declaration}} = \underbrace{\texttt{new Buggle()}}_{\text{constructor method}}}^{\text{assignment statement}};$$



Four big ideas 2-18

Behind the curtain

```
Buggle becky = new Buggle();
```

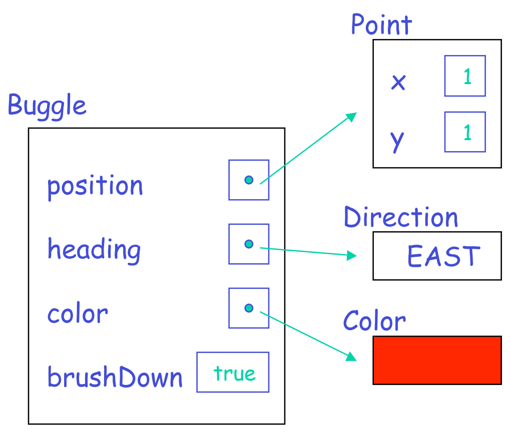


A class is described by

instance variables
describe the properties of each class instance;

instance methods
are the messages to which an instance of the class can respond;

constructor method(s)
create new instances of the class.



Buggle code

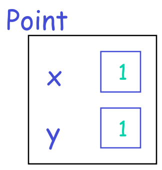
```
class Buggle
{

   private Point position;                          instance
   private Direction heading;                       variables
   private Color color;
   private boolean brushDown;

   public Buggle() { ... }                          constructor
                                                     method

   public void forward() { ... }
   public void left() { ... }
   public Color getColor() { ... }                  instance
   public Point getPosition() { ... }               methods
   public void setColor(Color c) { ... }
   public void setPosition(Point p) { ... }
   ...
}
```

Are there classes other than Buggles?

- Yep!
- Java is an object-oriented language, which means that programs construct and manipulate objects inside the computer that represent objects in the real world.
- Every object belongs to a class.*

*And these classes are designed modularly. That's where the real power of oops programming lies.

Points are objects*



```
class Point
{

    public int x;
    public int y;

    public Point(int x, int y) { ... }
    public Point() { ... }
    public Point(Point p) { ... }


    public void setLocation() { ... }
        ...
}
```

instance variables

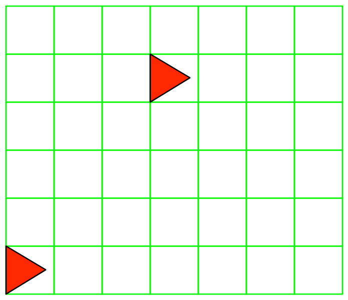
constructor methods

instance methods

*The Point class represents a location in two-dimensional (*x*, *y*) coordinate space.

Beam me up Scotty

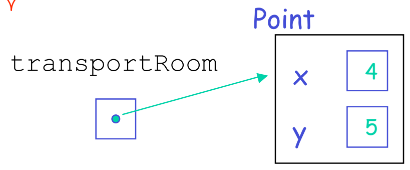
```
import java.awt.*;
public class EnterpriseWorld extends BuggleWorld
{
    public void run()
    {
        Buggle kirk = new Buggle();
        Point transportRoom = new Point(4,5);
        kirk.setPosition(transportRoom);
    }
}
```



Instance variables

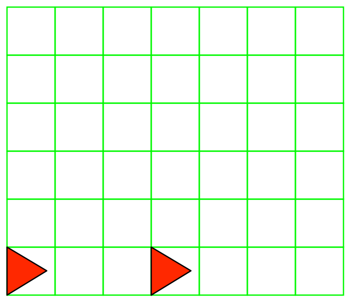
```
public class EnterpriseWorld extends BuggleWorld
{
    public void run()
    {
        …
        Point transportRoom = new Point(4,5);
        …
    }
}
```

Creates a new instance of the class Point named `transportRoom`



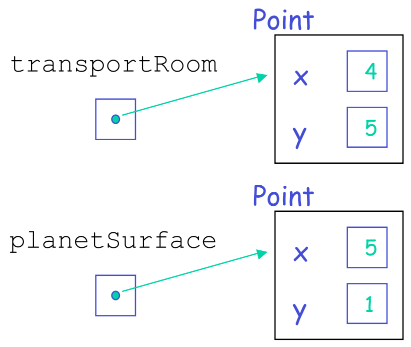
A Klingon trick

```
public class EnterpriseWorld extends BuggleWorld
{
    public void run()
    {
        Buggle kirk = new Buggle();
        Point transportRoom = new Point(4,5);
        transportRoom.y = 1;  // surprise kirk
  →     kirk.setPosition(transportRoom);
    }
}
```

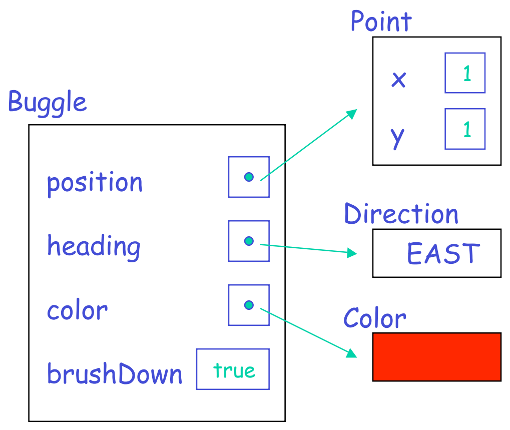


Another pair of instance variables

```
public class EnterpriseWorld extends BuggleWorld
{
    public void run()
    {    …
         Point transportRoom = new Point(4,5);
         Point planetSurface = new Point(5,1);
         …
    }
}
```

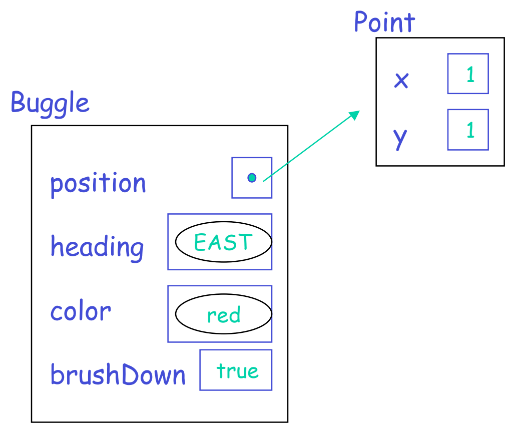


Objects inside of objects

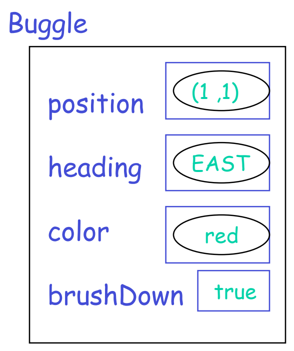


*What about contents of variables *x* and *y* inside of a Point object? Are they objects too?

A notational convenience



Sometimes we abbreviate further still


Buggle
position
(1 ,1)
heading
EAST
color
red
brushDown
true