# Introduction to Java

February 15–16, 2012
CS 60: Principles of Computer Science

**Assignment 4 due Monday February 20: Prolog Puzzles**

**Also on Monday, February 20: Guest Lecture by Paul Ruvolo (HMC '03)**

# A Very Short History of Java

Oak (1991) ⟶ Java (1996)

Key attributes

- ✓ Portability ("write once, run everywhere")
  - Java™ Language Specification
  - Java Virtual Machine
- ✓ Safety
  - (Compile time) Type checking
  - JVM Verifier
  - Garbage Collection
- ✓ Familiarity
  - Imperative language
  - C-like syntax
  - C++-like classes

# What's a Program?

- ✓ **Python**: A collection of variable definitions, function definitions, and class definitions (and expressions to evaluate).
- ✓ **Racket**: A collection of variable definitions and function definitions (and expressions to evaluate).
- ✓ **Prolog**: A collection of facts and rules.
- ✓ **Java**: A collection of class definitions. (That's it!)

Point.java

```
class Point
{
    ....class contents....
}
```

# Compiling and Running Programs

## Command-line (Terminal/cmd) Approach

*write* `Point.java` *using a text editor*

```
% javac Point.java
```

*compiles to bytecode file* `Point.class`

```
% java Point
```

*runs* `Main` *in* `Point.class`

# JVM Bytecode Sample

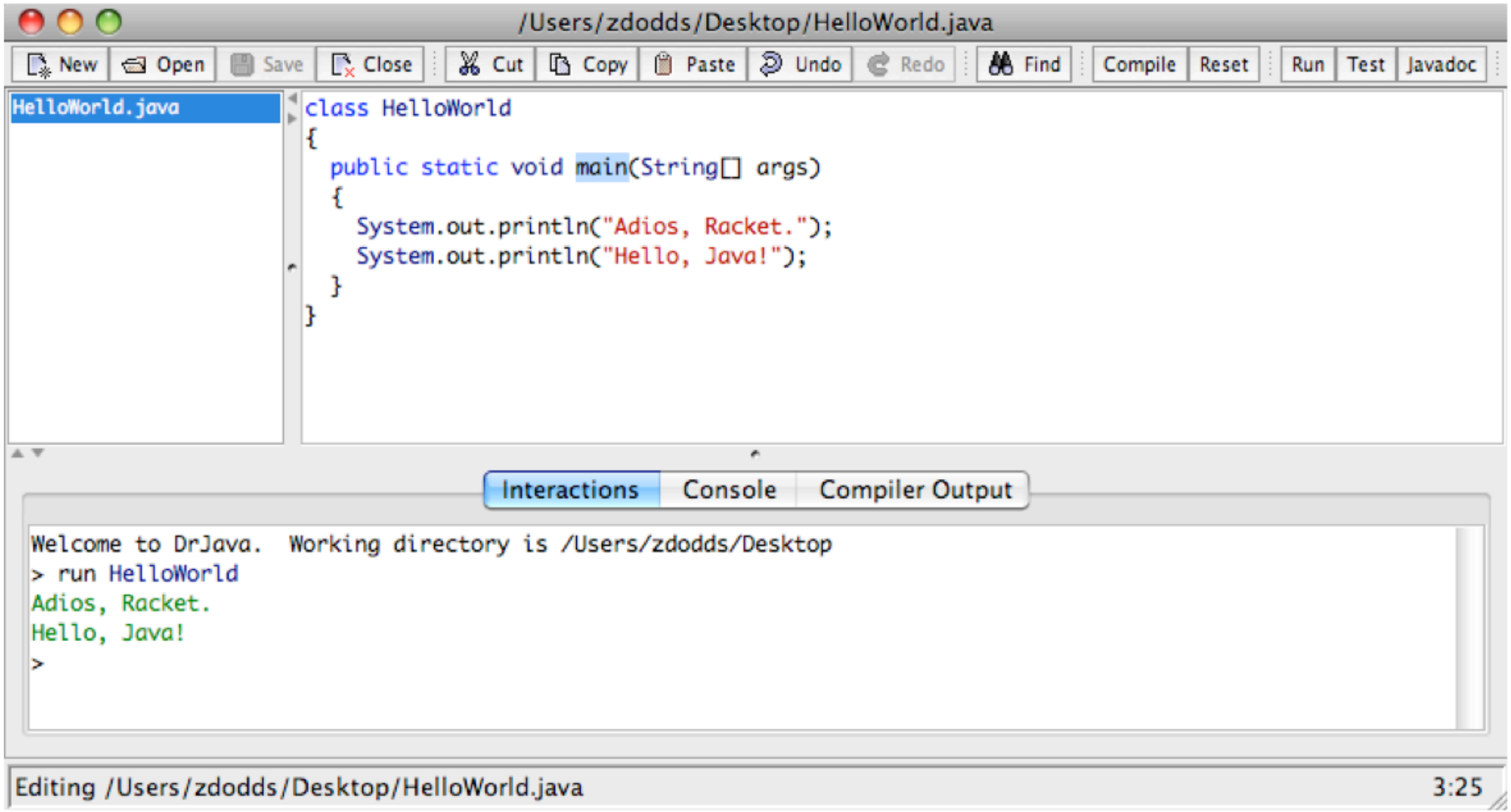
```
class ItsAFac
{
  public static double fac(int N)
  {
      if (N<2)
      {
          return 1.0;
      }
      else
      {
          return N*fac(N-1);
      }
  }

  public static void main(String[] args)
  {
    int x = 4;
    x = x+1;

    double solution = fac(x)/2.0;
    System.out.println("solution: " + solution);
  }
}
```

```
public static double fac(int);
 Code:
  0:  iload_0
  1:  iconst_2
  2:  if_icmpge     7
  5:  dconst_1
  6:  dreturn
  7:  iload_0
  8:  i2d
  9:  iload_0
  10: iconst_1
  11: isub
  12: invokestatic   #2; //Method fac;
  15: dmul
  16: dreturn
```

```
public static void main(java.lang.String[]);
 Code:
  0:  iconst_4
  1:  istore_1
  2:  iload_1
  3:  iconst_1
  4:  iadd
  5:  istore_1
  6:  iload_1
  7:  invokestatic   #2; //Method fac:(I)D
  < more -- too much to fit here! >
```

# Compiling and Running Programs

## IDE Approach




We'll be using DrJava, but you're free to use any IDE.

# Data in Java

Primitive Data (*not* objects!)

- ✓ Integer: `int` (and `short` and `byte` and `long`)
- ✓ Floating-Point: `double` (and texttttfloat)
- ✓ Booleans: `boolean`
- ✓ Unicode characters: `char`

Objects: Data ("fields") + Code ("methods")

- ✓ `Object`
- ✓ `String`
- ✓ arrays (fixed-size)
- ✓ `Vector` (growable)
- ✓ "Wrapper classes": `Integer`, `Double`, ...
- ✓ user-defined classes

# Objects vs. Classes

Objects

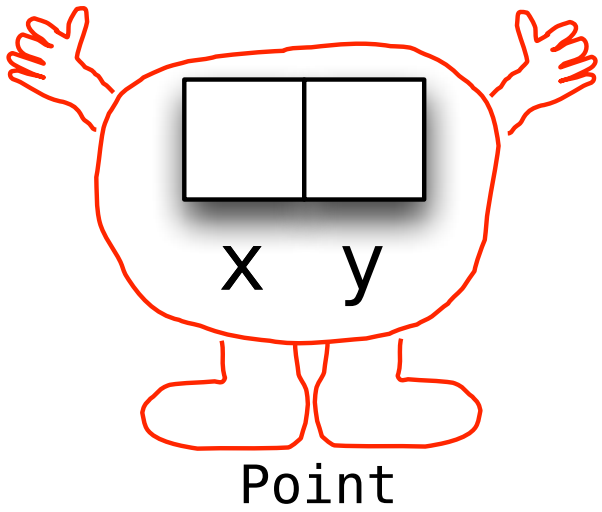
- ✓ Are created at run-time
- ✓ Contain data
- ✓ Have associated code (methods)
- ✓ Have an identity (address in memory)

Classes:

- ✓ Are described at compile-time
- ✓ Provide a "pattern" for describing/creating objects
- ✓ May include other related bits of code and data.

# Point OBJECTS

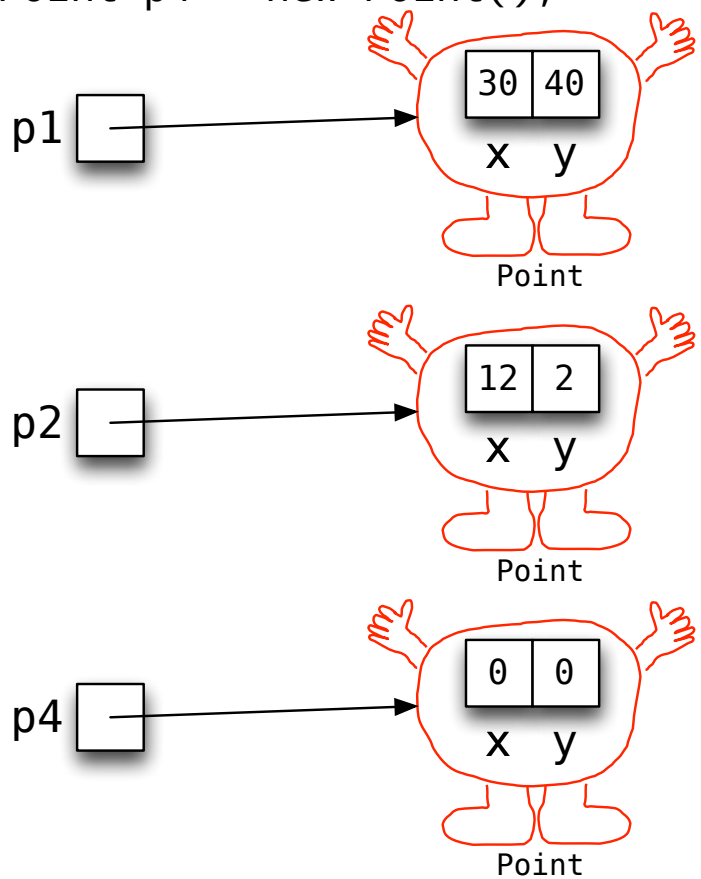
```
class Point extends Object
{
  private double x;
  private double y;
  ...
}
```



# Constructors

```
class Point extends Object
{
  ...
  public Point(double x_in,
               double y_in)
  {
    this.x = x_in;
    this.y = y_in;
  }

  public Point()
  {
    this.x = 0.0;
    this.y = 0.0;
  }
  ...
}
```

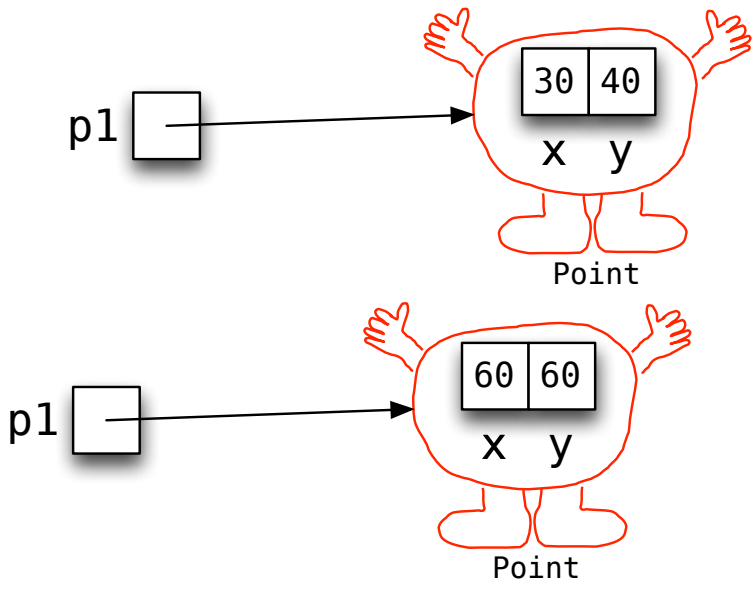
```
Point p1 = new Point(30,40);
Point p2 = new Point(12,2);
...
Point p4 = new Point();
```



# Methods: nudgeBy

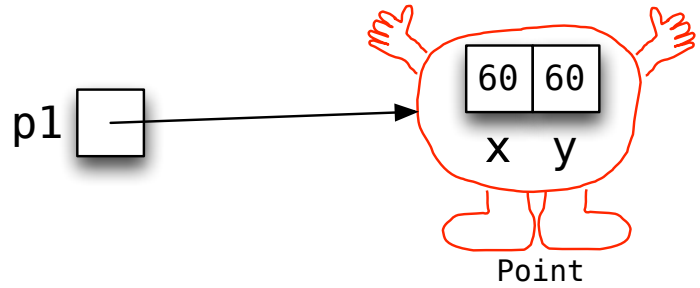
```
class Point extends Object
{
  ...
  public void nudgeBy(double delta_x,
                      double delta_y)
  {
    this.x = this.x + delta_x;
    this.y += delta_y;
    return;
  }
  ...
}
```

```
Point p1 = new Point(30, 40);

p1.nudgeBy( 30, 20 );
```

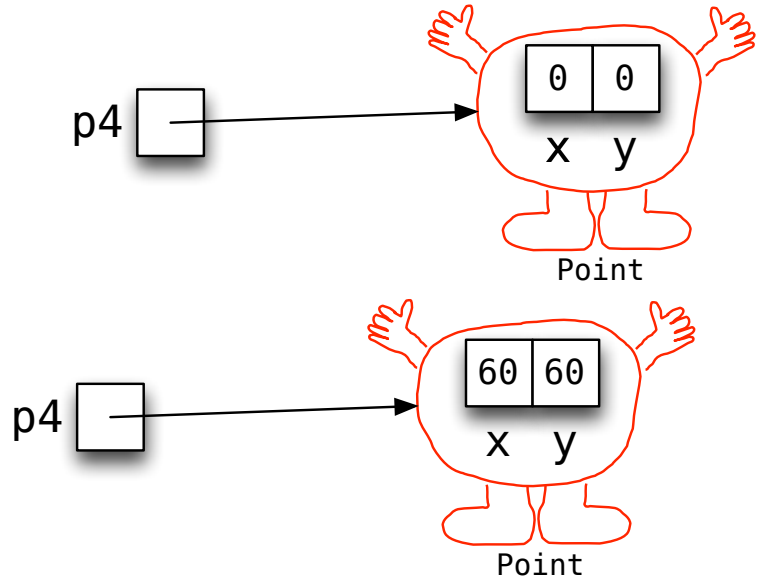


# Methods: equals

```
class Point extends Object
{
  ...
  public boolean equals(Point other)
  {
    if ( this.x == other.x  &&
         this.y == other.y ) {
      return true;
    } else {
      return false;
    }
  }
}
```




```
Point p4 = new Point();
p4.nudgeBy( 60, 60 );
```

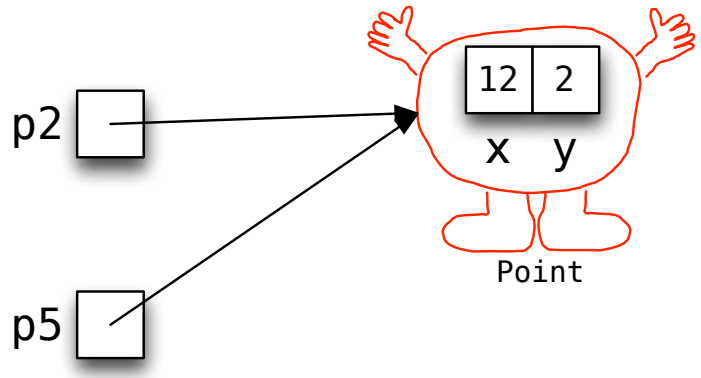



What are

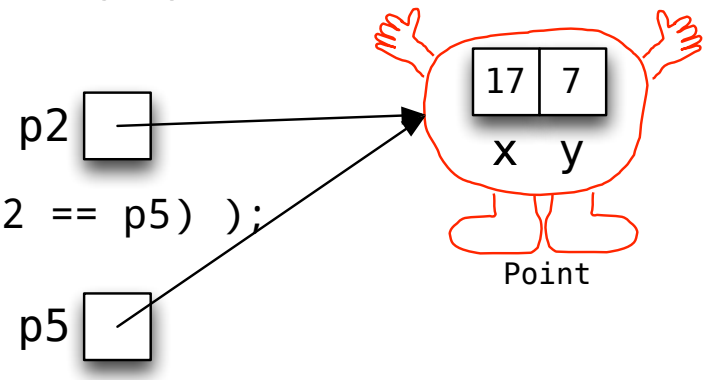
# Assigning Objects

```
Point p2 = new Point(12, 2);
...
Point p5 = p2;
System.out.println("Before nudge:");
System.out.println("p2 is " + p2);
System.out.println("p5 is " + p5);

p2.nudgeBy(5, 5);

System.out.println("After nudge:");
System.out.println("p2 is " + p2);
System.out.println("p5 is " + p5);
System.out.println("p2 == p5 is " + (p2 == p5) );
```

Before nudging:




After nudging:



# `static` METHODS

Sometimes we want an "ordinary" function, not associated with any particular object.

But everything in Java has to live in some class...

Solution (Hack?): Stuff these functions into a class, labeled `static`.

```
static public Point add( Point p1,
                         Point p2 )
{
   return new Point( p1.x + p2.x,
                     p1.y + p2.y );
}
```

```
Point p1 = new Point(30, 40);
Point p2 = new Point(12,  2);
     ...
p1.nudgeBy( 30, 20 );
     ...
Point p3 = Point.add( p1, p2 );
```