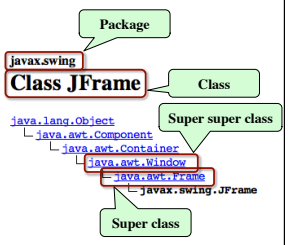
A Interesting Challenge

- How do we add new methods to **Rhino**?
  - Open up the .java file and add them!
- Java has a lot "built-in" classes
  - **Examples:** String, Vector, JFrame
- What if we want to add methods to these?
  - We cannot access the .java file (where is it???)
- But we can create a **subclass**
  - A new class with all fields, methods of the "parent"
  - Class also contains anything new we want to add

Subclasses in the Java API

- Subclassing creates a hierarchy of classes
  - Subclass has a super class or "parent" class
  - That parent may have a super class as well
- Explicit in the Java API
  - API does not respecify **inherited** methods
  - Often have to go to super class for specification



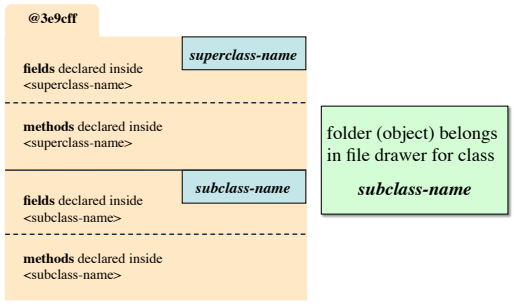
Class Definition REVISITED

- Describes the format of a folder (instance, object) of the class.

```
/**
 * Description of what the class is for
 */
public class <class-name> extends <super-class> {
      declarations of fields and methods (in any order)
}
```

- Class <class-name> has all methods and fields of its parent
  - We say that it **inherits** them
- Also has any new fields or methods declared inside of it

Folder Analogy and Subclasses



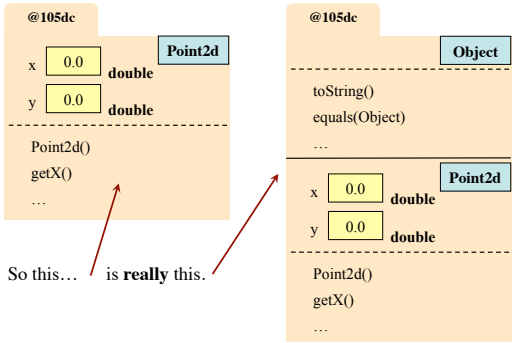
Subclassing a JFrame

```
/** Description of what the class is for… */
public class SquareJFrame extends JFrame {
   /** Set the height of the window to the width */
   public void setHeightToWidth() {
      setSize(getWidth(),getWidth());
   }
   /** Yields: the area of the window */
   public int area() {
      return getWidth()*getHeight();
   }
   …
}
```

Inherited method which is used as a helper method

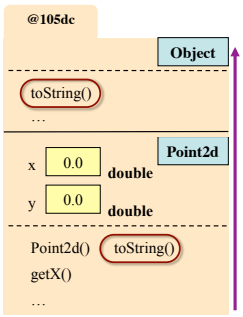
folder (object) belongs in file drawer for class SquareJFrame

Object: The Superest Class of All



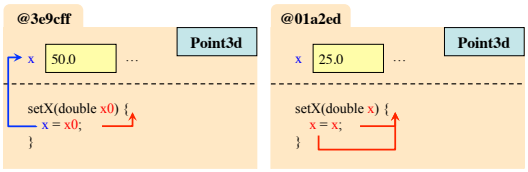
The Bottom-Up Rule

- Which toString() is called?
  - Work the way up from the bottom of the folder.
  - Find the first method header that matches
  - Use the definition from the .java file for that class
- New method definitions **override** those of super class



Inside-Out Rule (See p. 83)

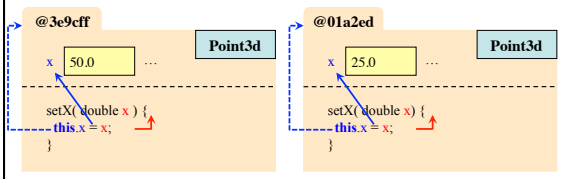
- Parameter x0 is found in the frame for the method call. Exists temporarily
- Parameter x "blocks" (or **shadows**) the reference to the field x.



A Solution: `this`

`this` is a built-in "variable" that gives an object name

- In object (folder) **@3e9cff**, **this** refers to **@3e9cff**
- In object (folder) **@01a2ed**, **this** refers to **@01a2ed**



Keywords `this` and `super`

`this`

- Refers to the object name in scope box of the method call
- this.<field> is field in object
  - Example: this.x
- this.<method-call> calls a method in this object
  - Example: this.getX()
- this(<parameters>) calls a constructor
  - Example: this(0.0,0.0,0.0)

`super`

- Functions mostly the same as this (refers to object in scope)
- super.<method-call> calls a method in the superclass or even higher up!
- super(<parameters>) calls constructor of super class
  - Useful for initialization
  - Necessary if fields private

Using `this` as a Constructor

- Usage: `this(<params>)`
  - Looks for constructor with parameters of that type
  - Calls that constructor as a helper method
  - Can only do this inside another constructor
- This is why object name must be in the scope box
  - Else what is `this`?
  - `this` = name in scope box

```
public Point3d(double x0,
               double y0,
               double z0) {
  x = x0;
  y = y0;
  z = z0;
}

public Point3d() {
  // Uses other constructor.
  this(0.0,0.0,0.0)
}
```

Using `super` in a Constructor

```
public Employee(String n, int d) {
  name= n;
  start= d;
  salary= 50000;
}
```

```
public Executive(String n, int d,
                 double b) {
  super(n,d);
  bonus = b;
}
```

Must be first line