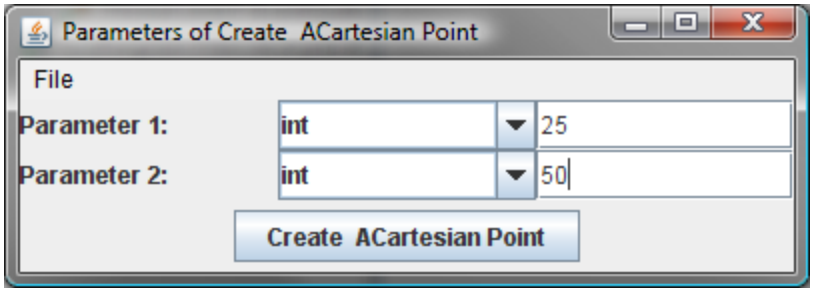
# Comp 110 and 401
# Class (Static) State

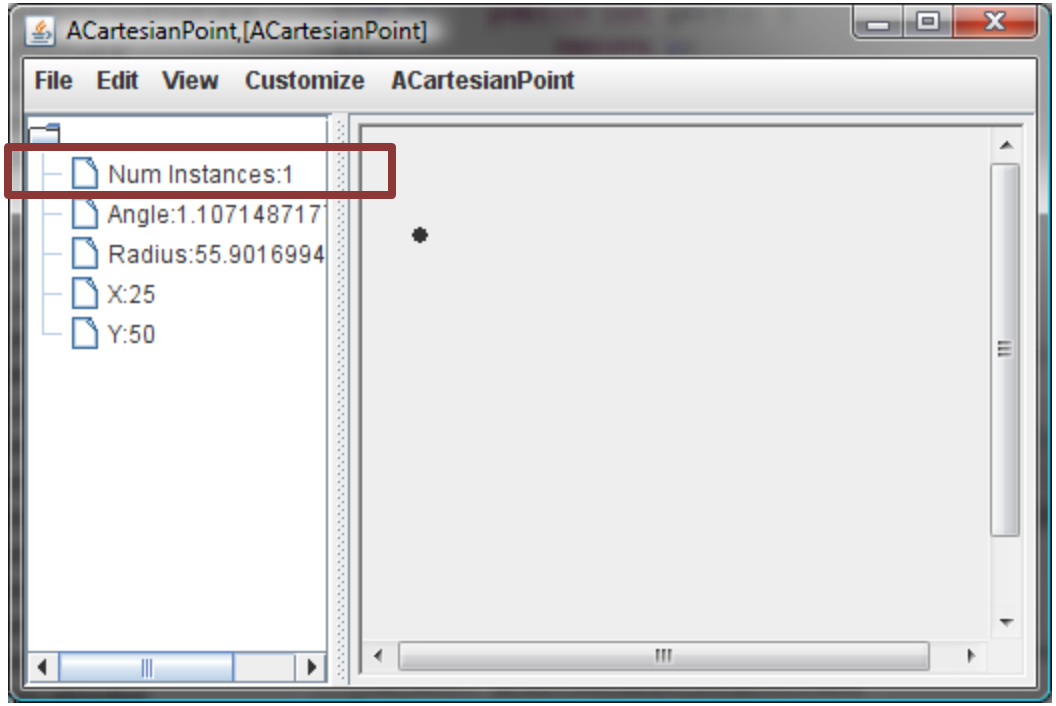
Instructor: Prasun Dewan

# Prerequisites

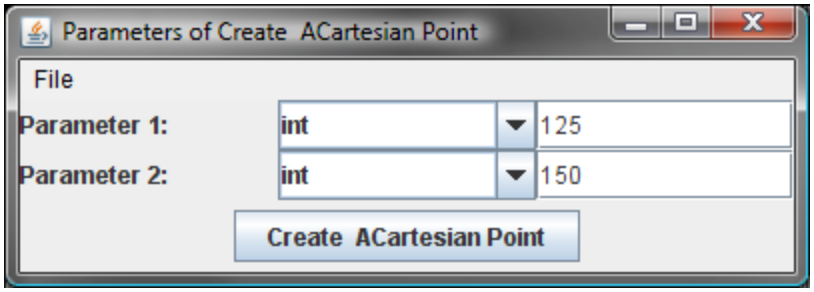
- State and Properties
- Interfaces

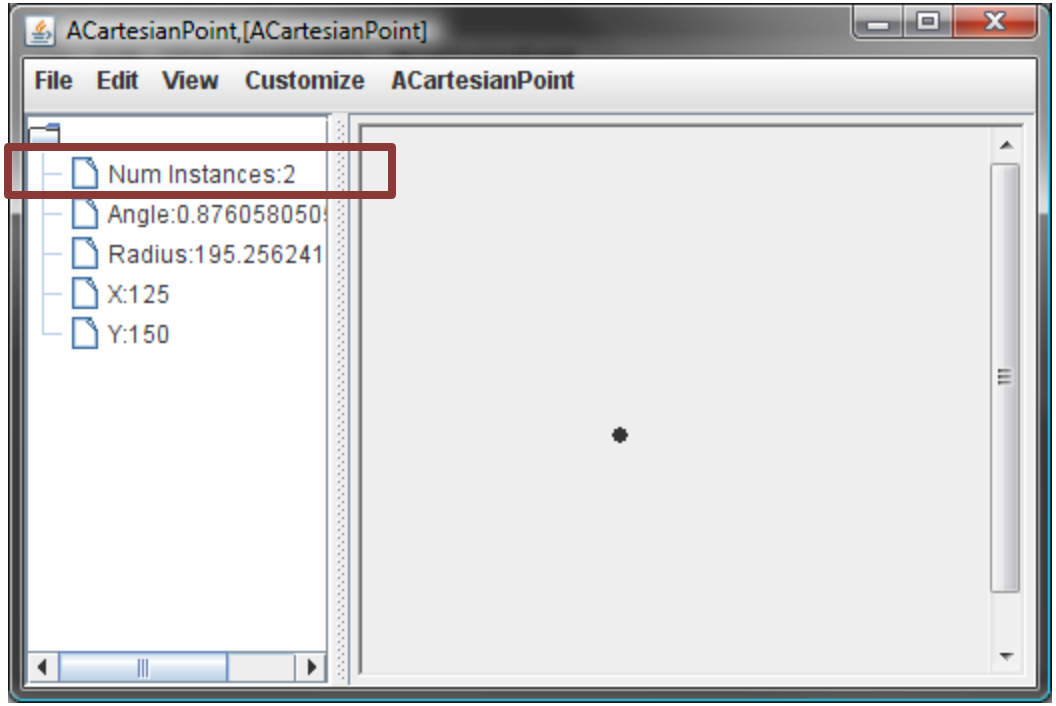
# Counting Instances of ACartesianPoint



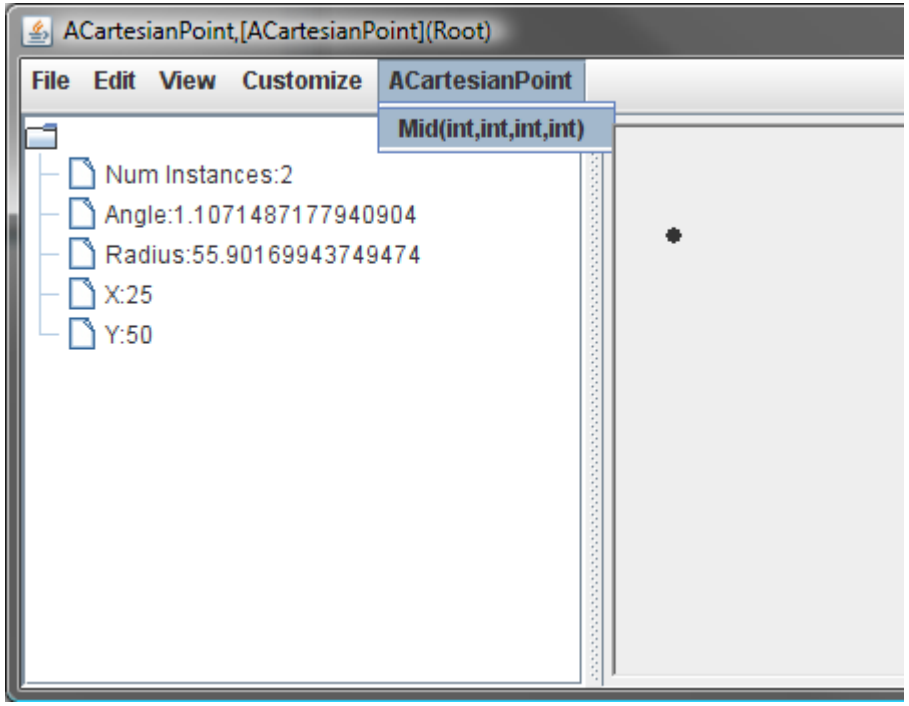



# Incremented Number of Instances

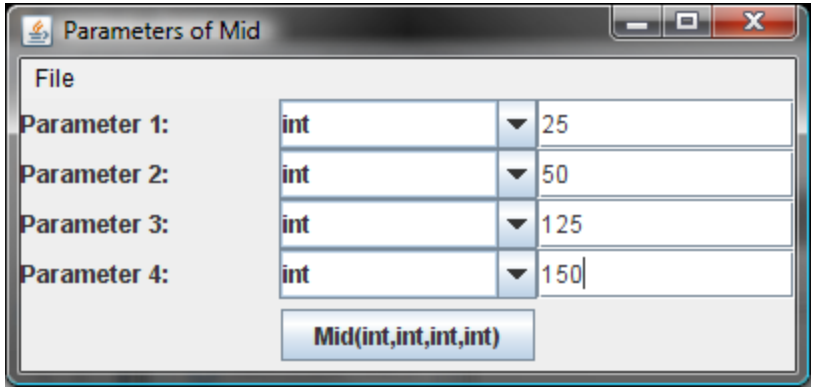


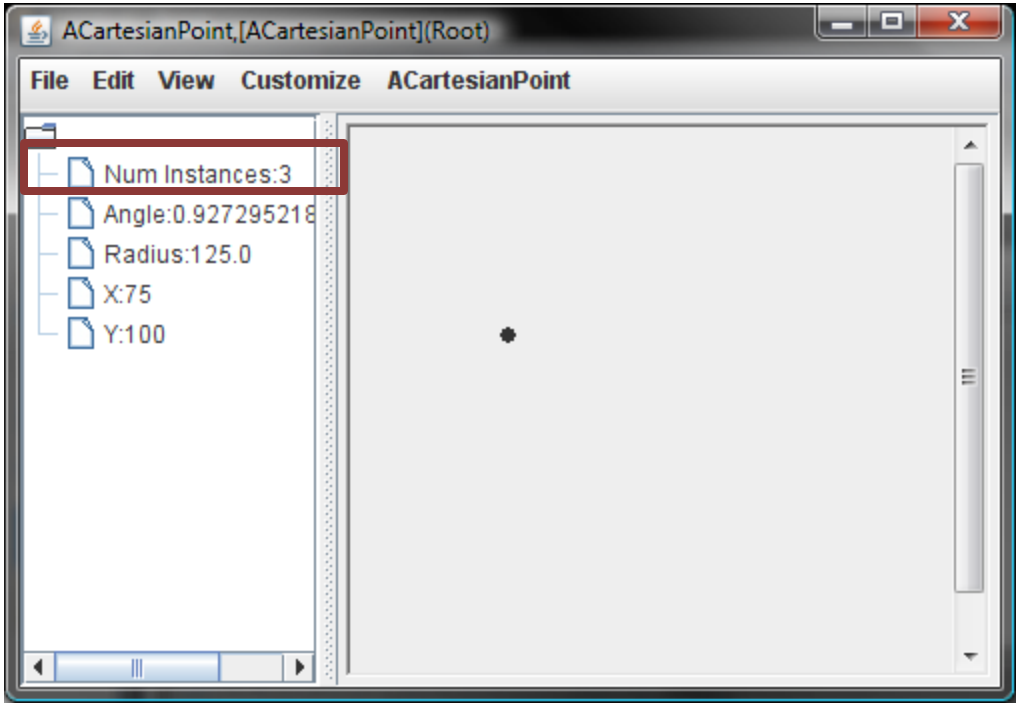



# Creating MidPoint



# Specifying Two End Points

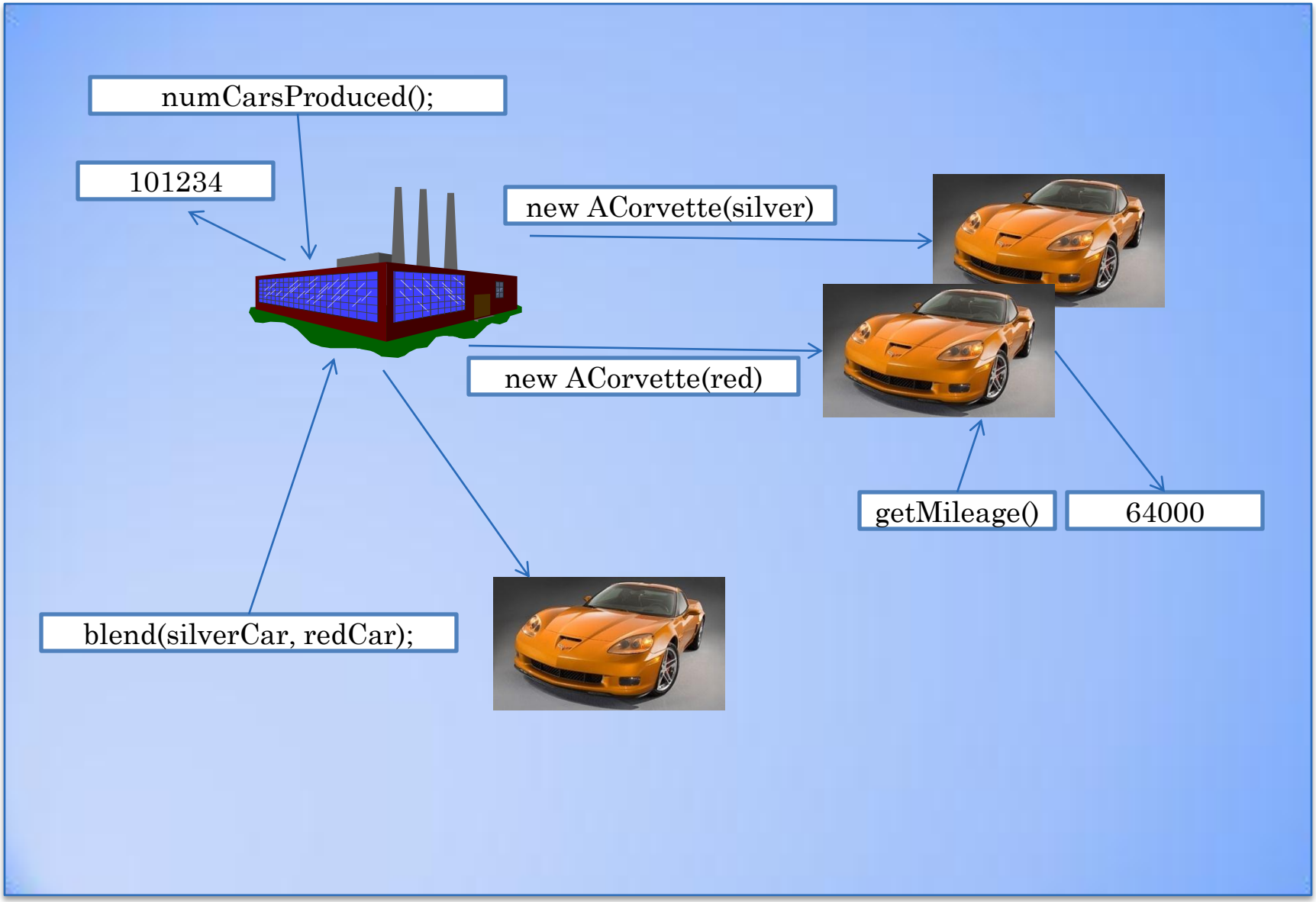


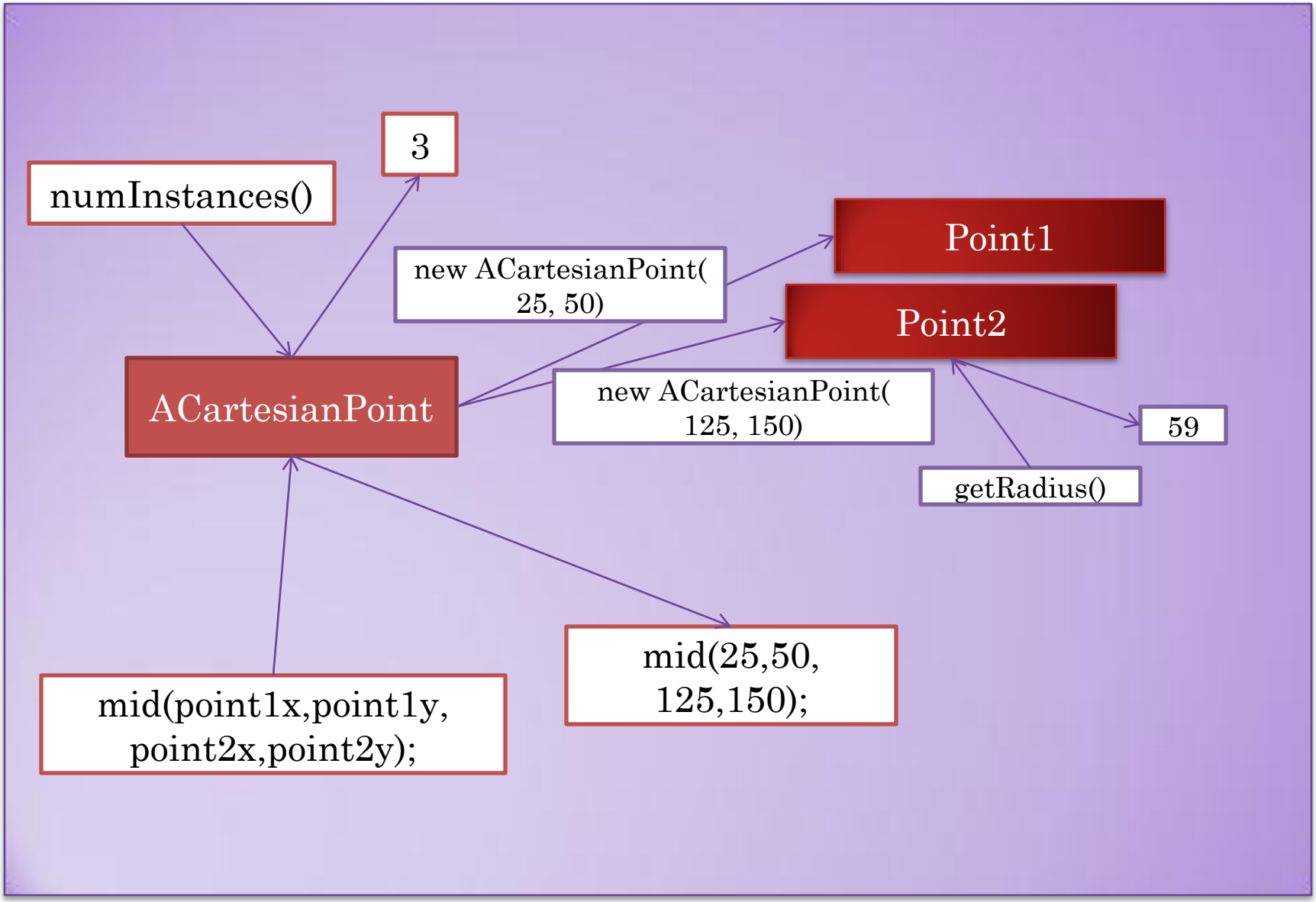
# Type of Methods

- Difference between method returning number of instances and other methods seen so far?
- Difference between mid and other methods seen so far?

# Real-World Analogy



# O-O World


3
numInstances()
new ACartesianPoint(
25, 50)
Point1
Point2
ACartesianPoint
new ACartesianPoint(
125, 150)
59
getRadius()
mid(25,50,
125,150);
mid(point1x,point1y,
point2x,point2y);

# Class Methods

- Methods can be invoked on class itself
- Called class or static methods
- Declared in class on which they are invoked
- Keyword **static** in header
- Accesses no instance variable

# Programmer-Defined Mid

Instance Method

```
public getRadius (){
        return  Math.sqrt(x*x + y*y);
}
```

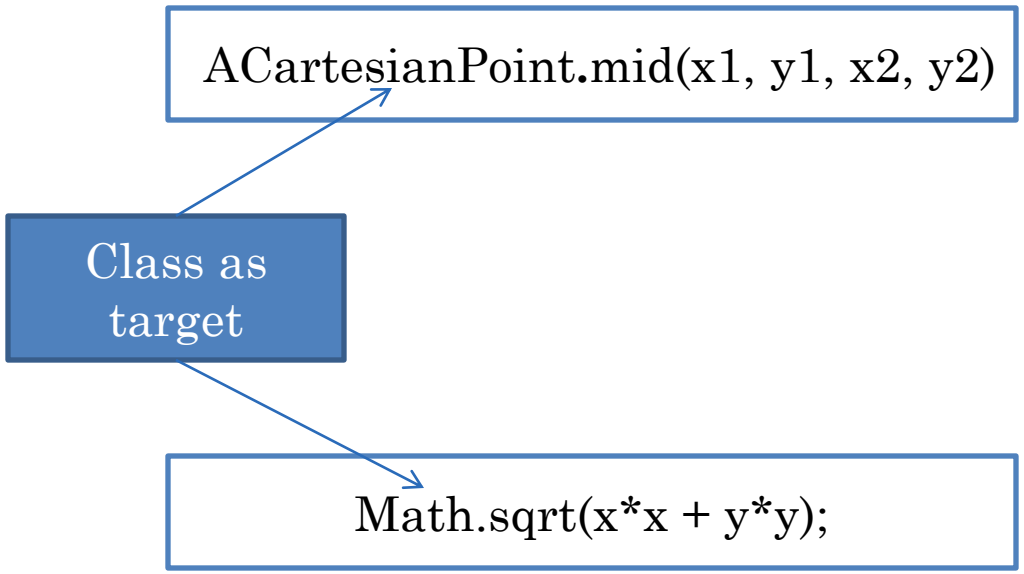
Access instance variables

Class Method

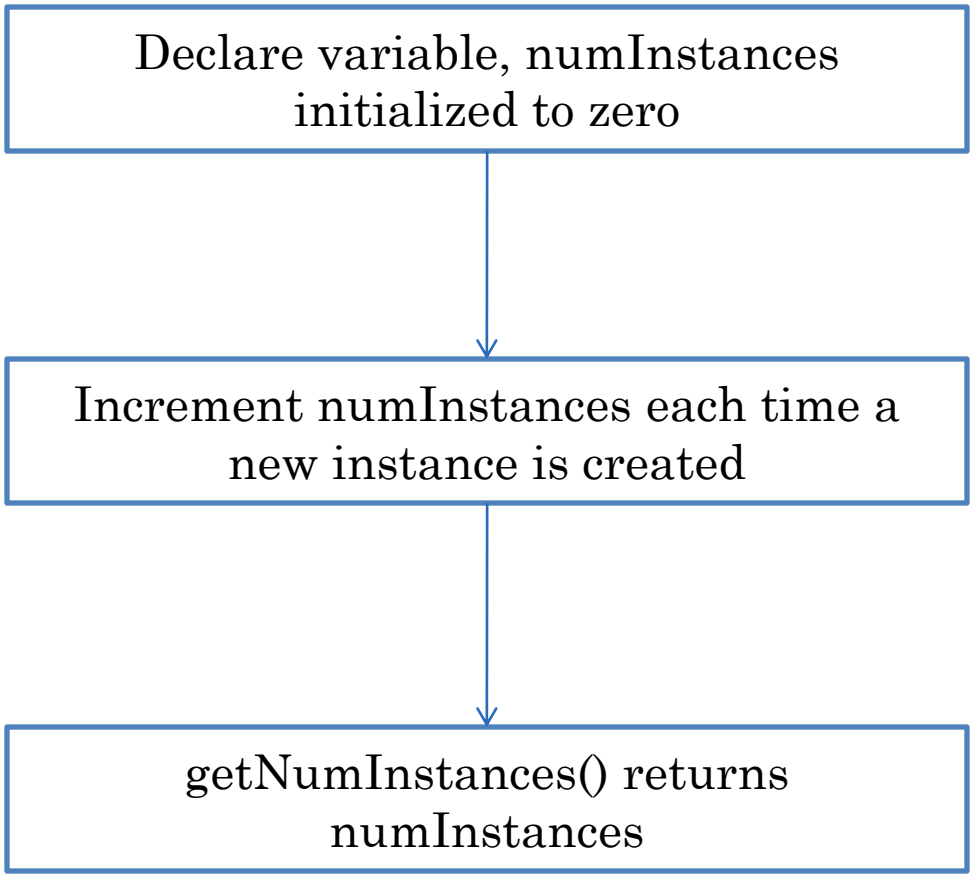
```
public static Point mid (int x1, int y1, int x2, int y2 ) {
      return new ACartesianPoint(x1 + (x2 - x1)/2, y1 + (y2 - y1)/2);
}
```

Access no instance variable

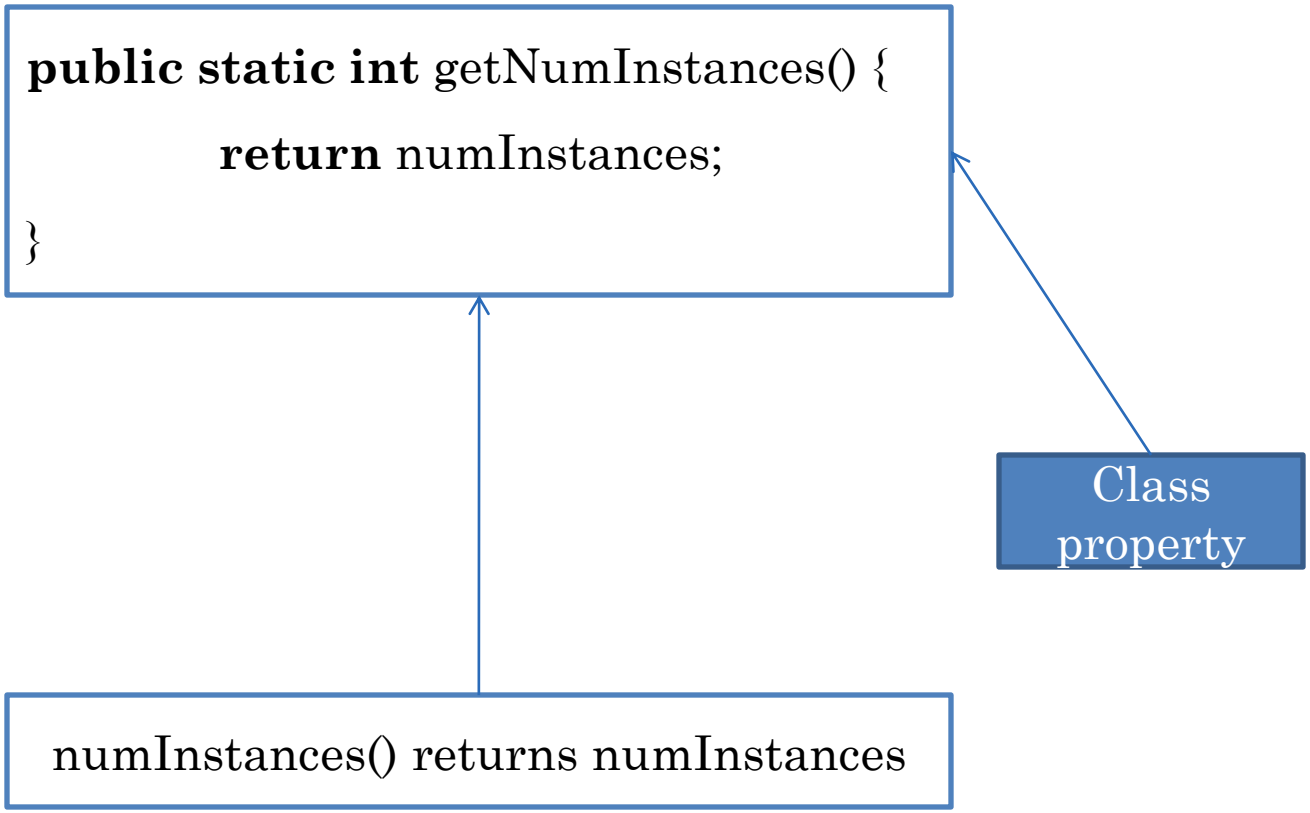
# External Call of Class Method



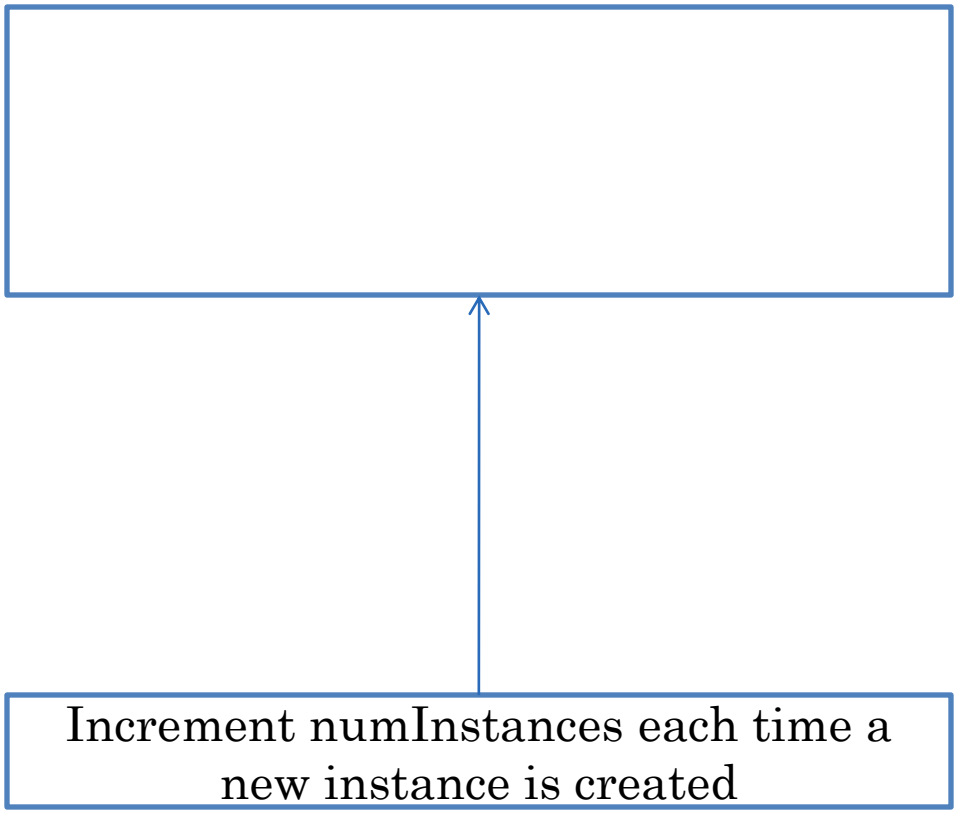
# numInstances Algorithm



# Returning numInstances



# Incrementing numInstances



# Incrementing numInstances (Solution)

```
public ACartesianPoint(int theX, int theY) {
        x = theX;
        y = theY;
        numInstances++;
}
```

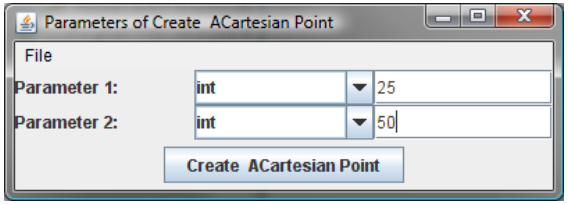
Increment numInstances each time a new instance is created

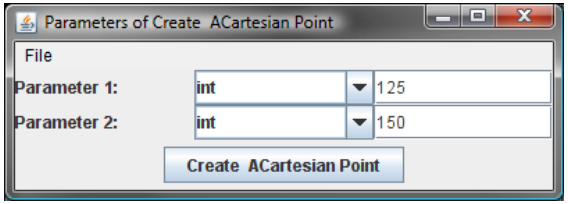
# Declaring numInstances

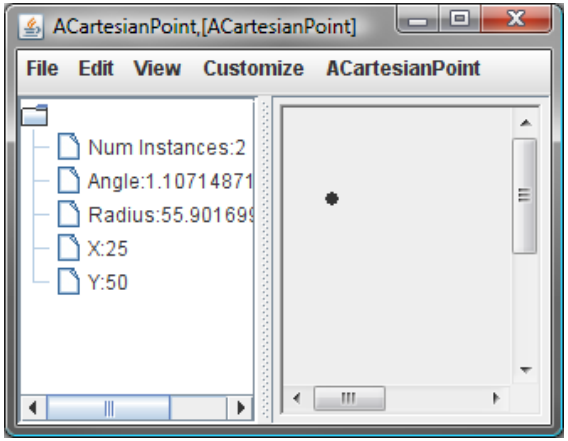
Declare variable, numInstances initialized to zero

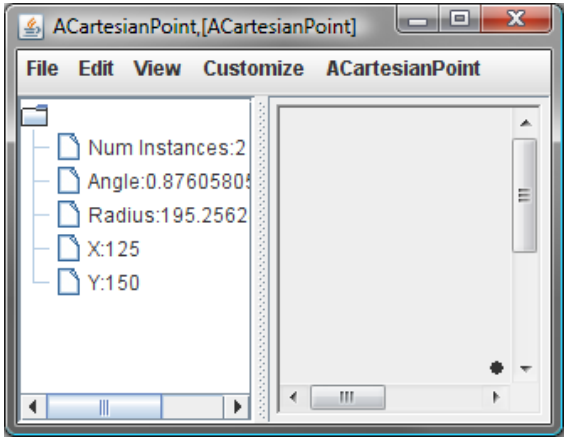
```
static int numInstances = 0;  // class variable
```

# Tracing Execution



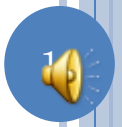




| variables | memory |
|---|---|
| numInstances; | 2 |
| x | 25 |
| y | 50 |
| x | 125 |
| y | 150 |

```
public ACartesianPoint(int theX, int theY) {
        x = theX;
        y = theY;
        numInstances++;
}
```

# Instance vs. Class Members

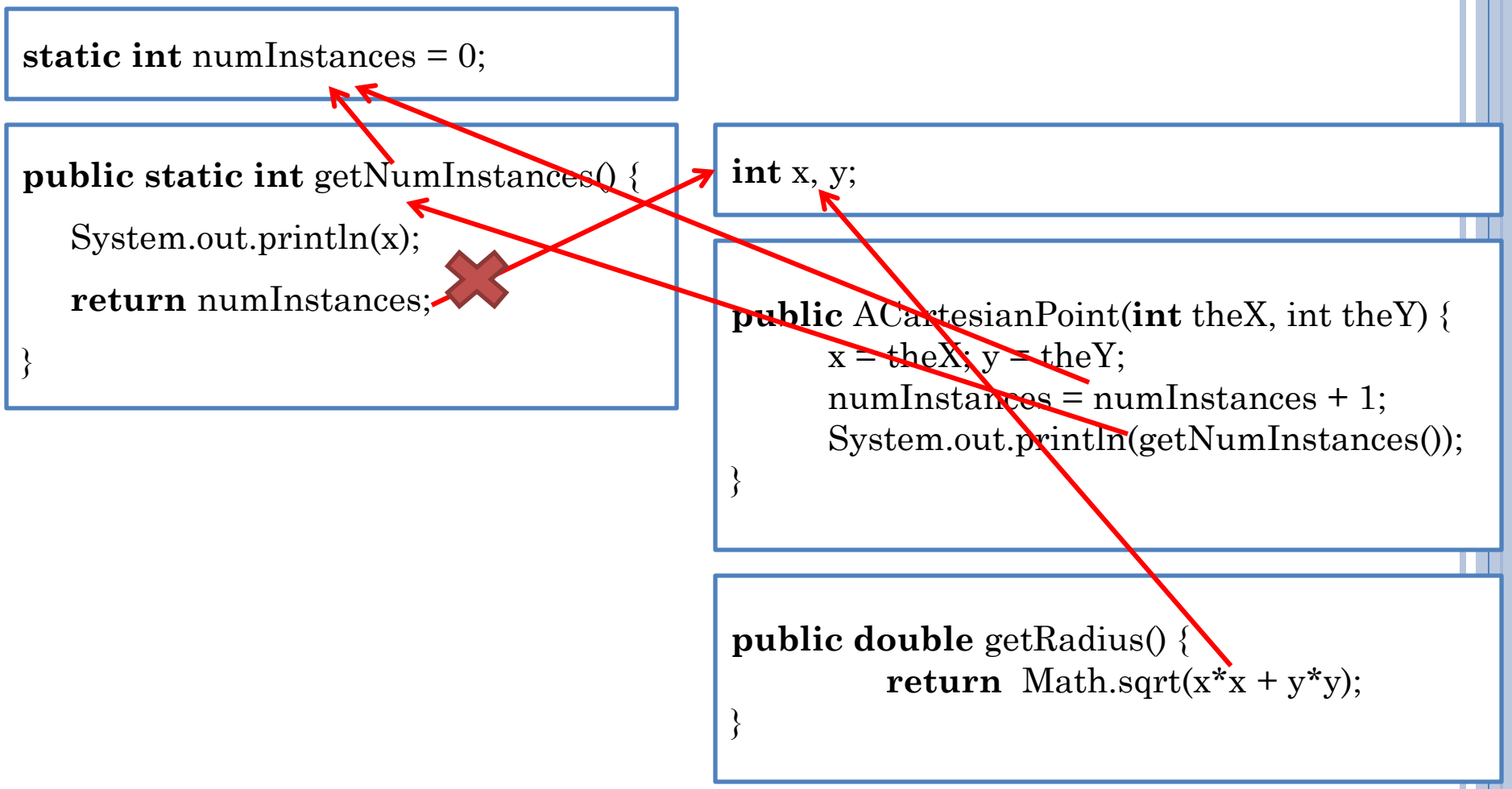
- Class Members
  - Class method
  - Class variables
- Instance Members
  - Instance methods
  - Instance variables



# Scope of Class and Instance Members

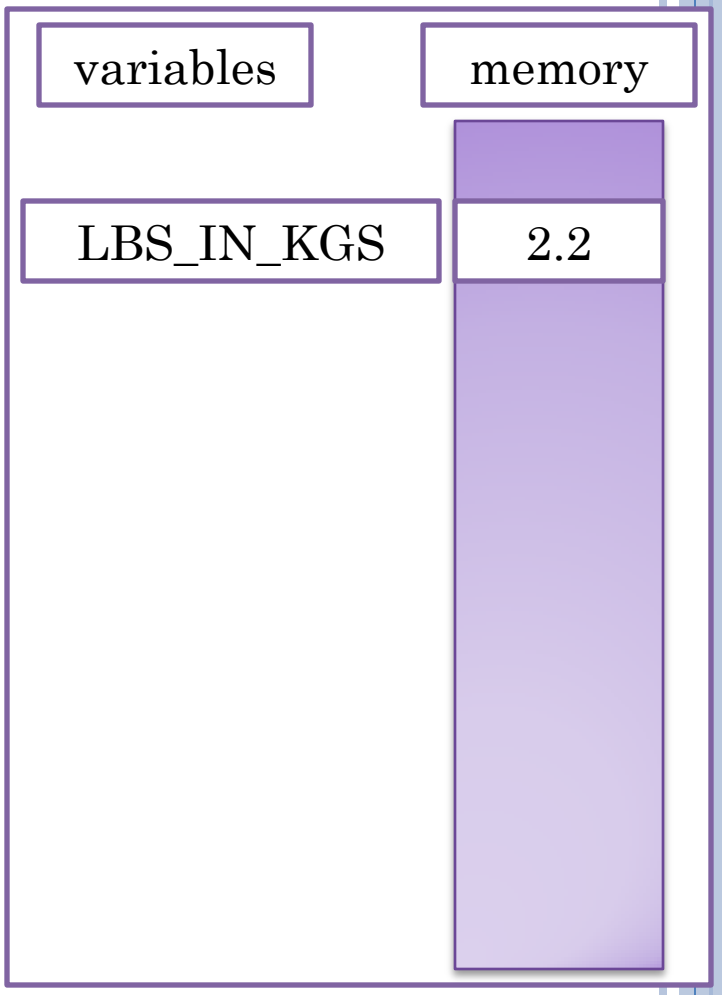
- Class Members
  - visible to other class members
  - visible to all instance members
    - class & instance methods can access class variables
    - class and instance methods can call class methods
- Instance Members
  - visible to other instance members
  - not visible to class members
    - which of (zero to many) copies of an instance variable should a class member refer to?

# Legal & Illegal Access



# Instance vs. Class Named Constant

```
public class AnotherBMISpreadsheet implements
BMISpreadsheet{
        double height, weight, bmi;
        ...
        public static final
                double LBS_IN_KG = 2.2;
        public static final
                double CMS_IN_INCH = 2.54;
        double calculateBMI() {
                return (weight/LBS_IN_KG) /
(height*CMS_IN_INCH/100*height*CMS_IN_INCH/10
0);
        }
}
```




```
new AnotherBMISpreadsheet()
new AnotherBMISpreadsheet()
AnotherBMISpreadsheet.LBS_IN_KGS;
```

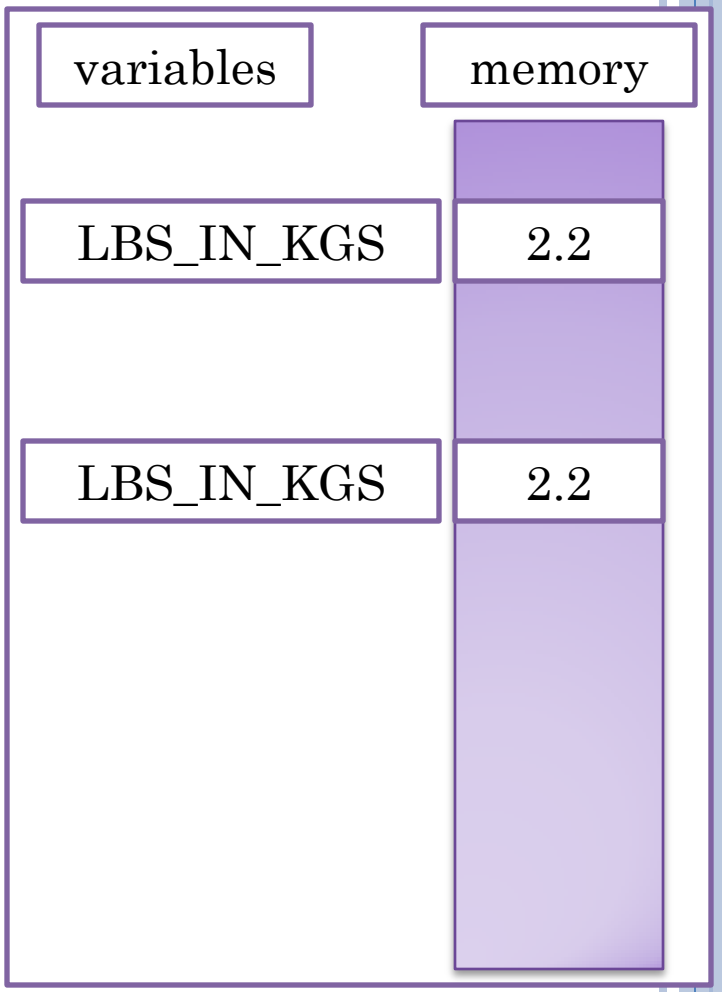
# Instance vs. Class Named Constant

```
public class AnotherBMISpreadsheet implements
BMISpreadsheet{
        double height, weight, bmi;
        ...
        public final
                double LBS_IN_KG = 2.2;
        public static final
                double CMS_IN_INCH = 2.54;
        double calculateBMI() {
                return (weight/LBS_IN_KG) /
(height*CMS_IN_INCH/100*height*CMS_IN_INCH/10
0);
        }
}
```

```
new AnotherBMISpreadsheet()
```

```
new AnotherBMISpreadsheet()
```

```
(new AnotherBMISpreadsheet()).LBS_IN_KGS;
```



# Class vs. Instance Constant

- Should be class constant
  - one copy
  - easy to refer (require no instance creation)
- Unless some good reason for hiding named constants from static methods

# Class vs. Instance Methods

Class Method

```
public static Point mid (int x1, int y1, int x2, int y2 ) {
        return new ACartesianPoint(x1 + (x2 - x1)/2, y1 + (y2 - y1)/2);
}
```

ACartesianPoint.mid(25, 50, 125, 150)

Math.round(5.7)

Accesses no instance variable

Instance Method

```
public Point mid (int x1, int y1, int x2, int y2 ) {
        return new ACartesianPoint(x1 + (x2 - x1)/2, y1 + (y2 - y1)/2);
}
```

(new ACartesianPoint(25, 50)).mid(25, 50, 125, 150)

(new Math()).round(5.7)

# Class vs. Instance Method

- Instance method has all the privileges of a class method
- Any class method can be made an instance method
- Bad style to have instance method that does not access any instance variable
  - They belong to the class
  - Violate least privilege principle
  - Require needless instantiation

# Class vs. Instance Constant/Method

- Named constants should be static
  - Unless some good reason for hiding named constants from static methods
- Methods not accessing instance variables should be static
  - Unless need to be listed in interface

# Class Member Restrictions

- Cannot define interfaces for classes
  - No conceptual reason why not
  - Non object oriented languages supported them
  - Class members go against the idea of OO programming (not instantiated) so treated as second class.
- Cannot use super in class methods
  - Can name super class directly
  - Non dynamic dispatch (discussed later)

# Static Example in Loan

- Revisit, through non-graphical objects, concepts illustrated in previous sections

# Instance Add

```
public Loan getTotalLoan(){
        return  houseLoan.add( carLoan);
}
```

```
public Loan add(Loan loan2) {
        return new ALoan(getPrincipal() + loan2.getPrincipal()));
}
```

# Class (Static) Add

Instance Method

```
public Loan getTotalLoan(){
        return ALoan.add(houseLoan, carLoan);
}
```

Accesses instance variables

Class Method

```
public static Loan add(Loan loan1, Loan loan2) {
    return new ALoan(loan1.getPrincipal() + loan2.getPrincipal()));
}
```

Access no instance variable

# Non-Polymorphic Methods

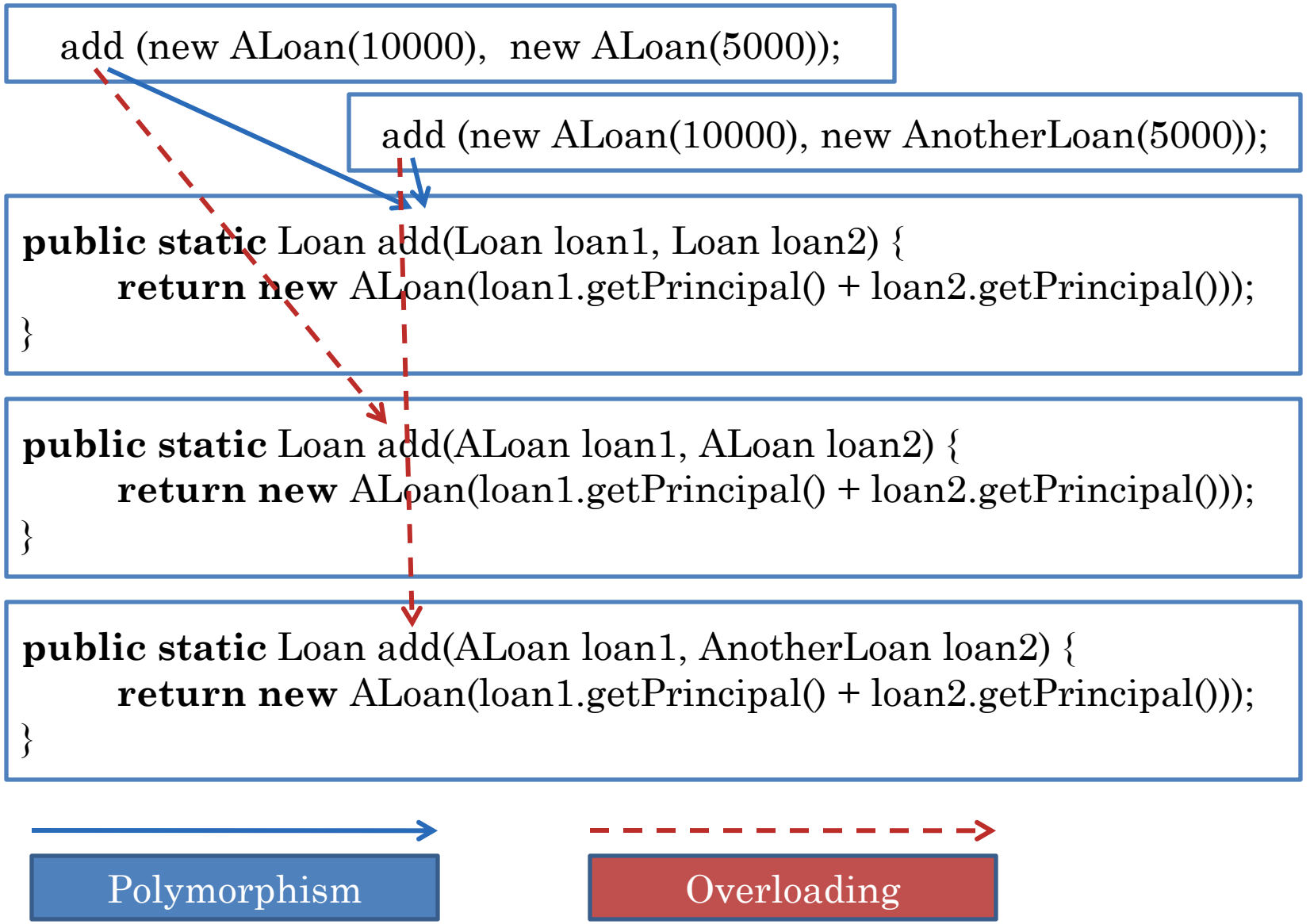
```
public static Loan add(ALoan loan1, ALoan loan2) {
        return new ALoan(loan1.getPrincipal() + loan2.getPrincipal()));
}
```

```
public static Loan add(AnotherLoan loan1, AnotherLoan loan2) {
        return new ALoan(loan1.getPrincipal() + loan2.getPrincipal()));
}
```

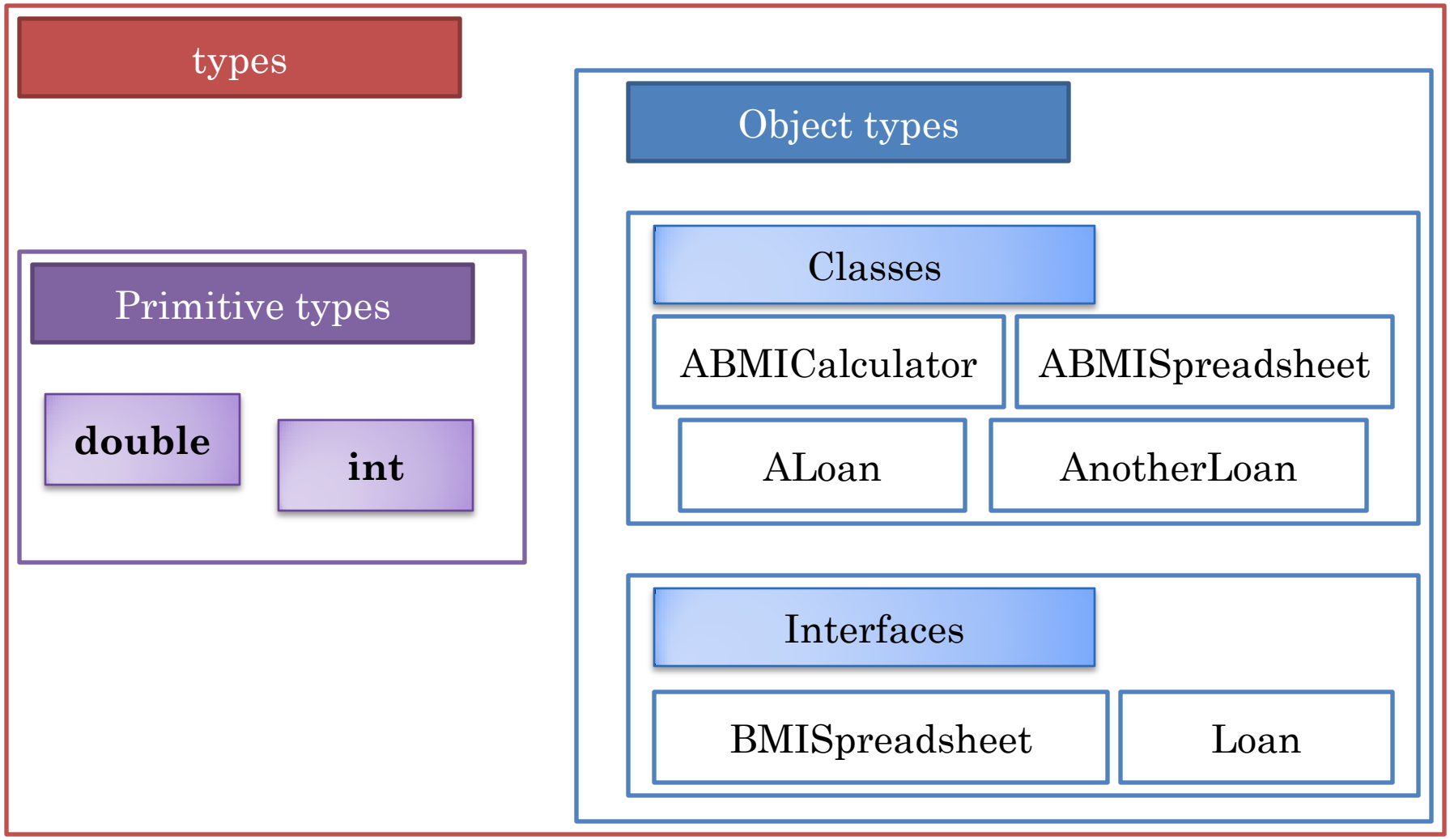
```
public static Loan add(ALoan loan1, AnotherLoan loan2) {
        return new ALoan(loan1.getPrincipal() + loan2.getPrincipal()));
}
```

Code duplication!

# Overloading vs. Polymorphism

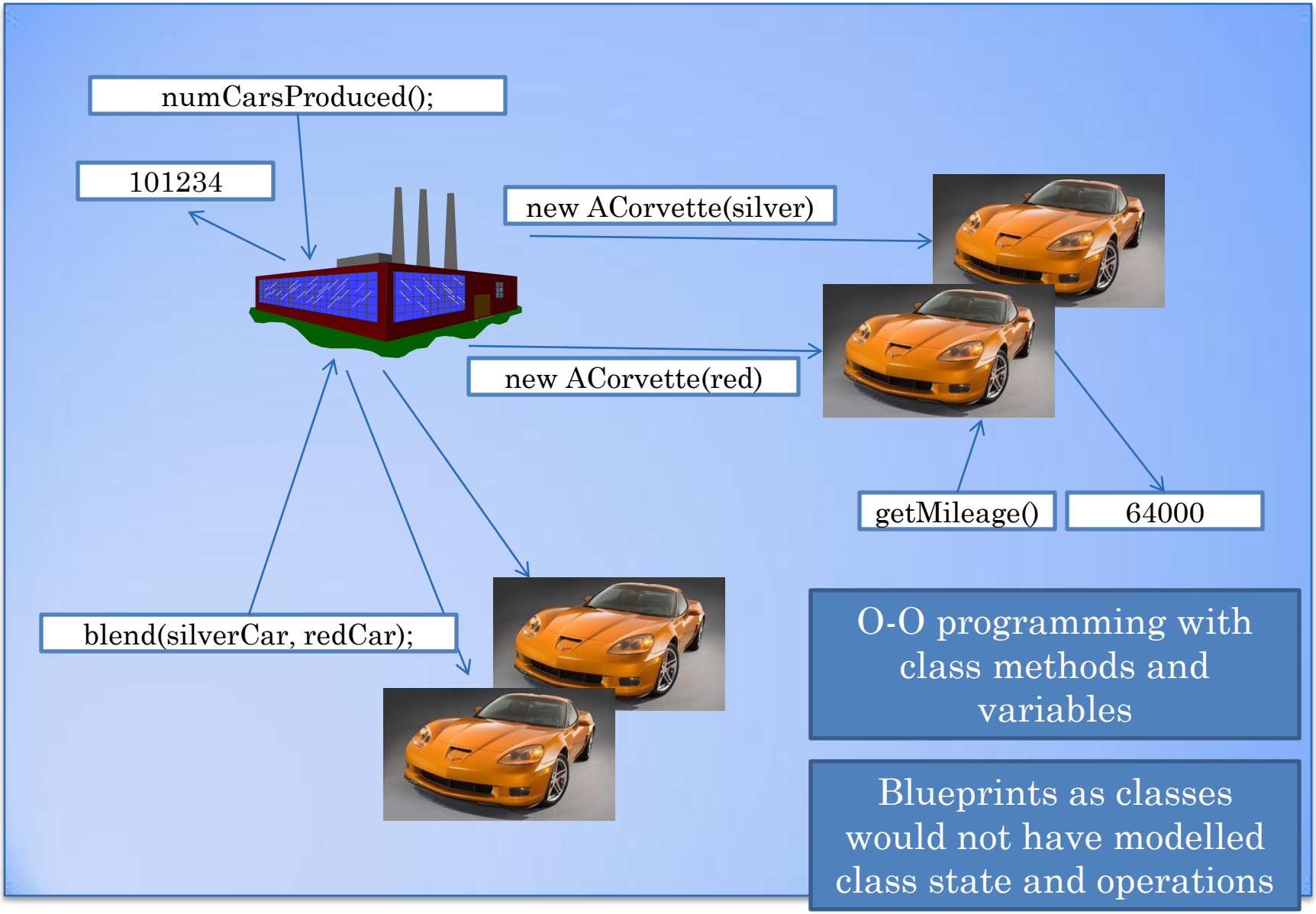


# Primitive vs. Object Types




type = set of operations

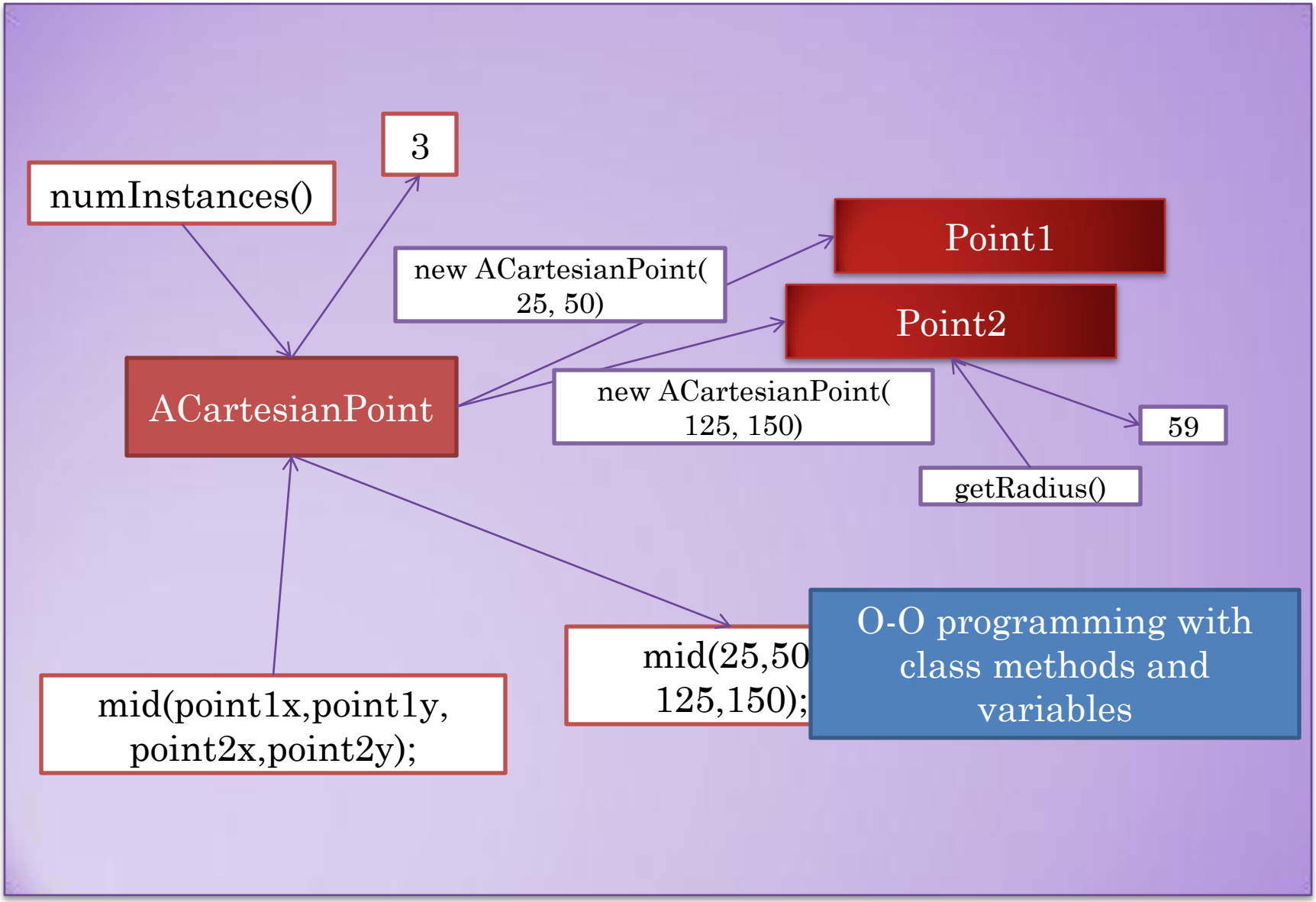
# Real-World Analogy (O-O Programming)



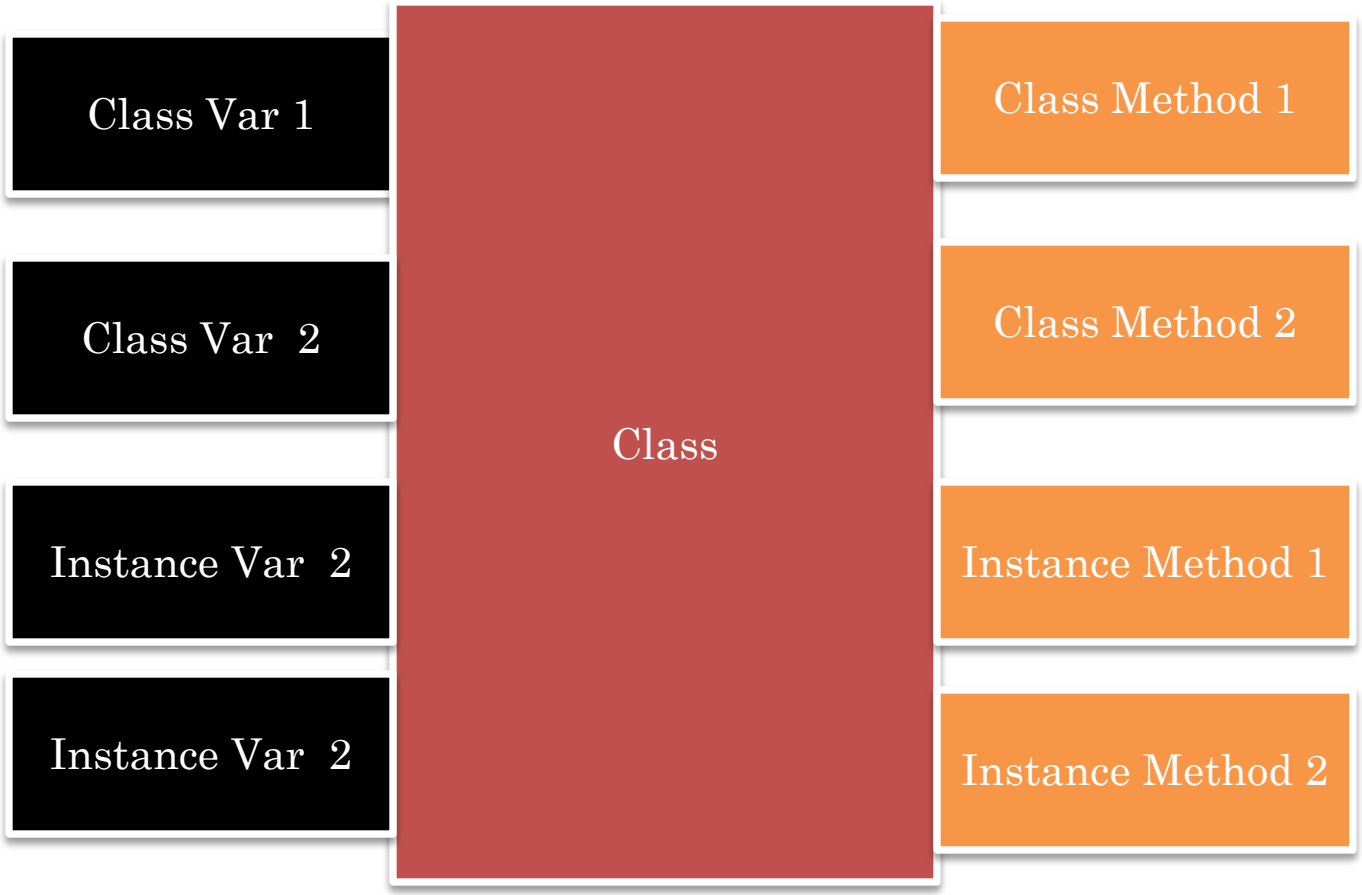

O-O programming with class methods and variables

Blueprints as classes would not have modelled class state and operations

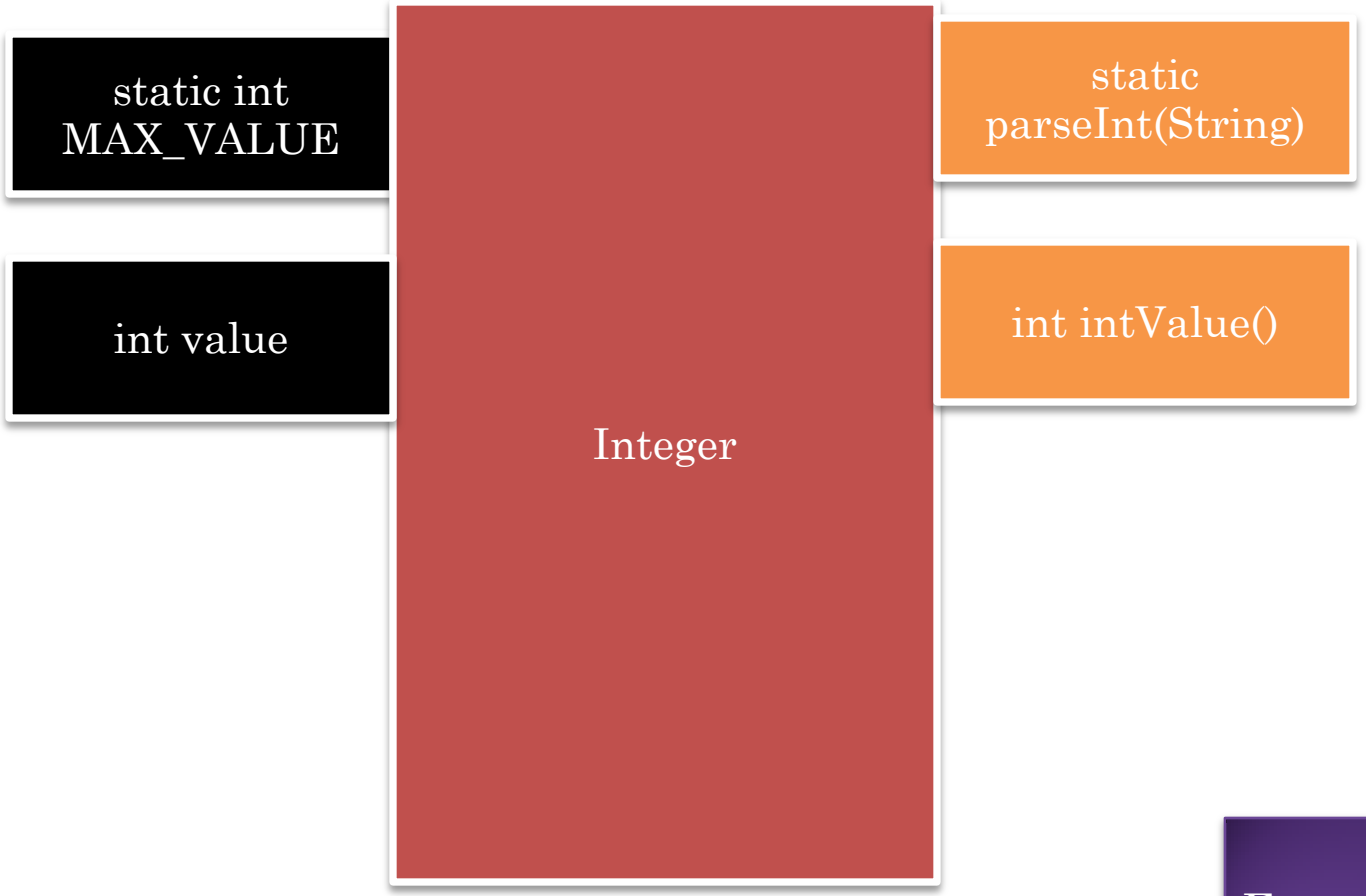
# O-O World


3
numInstances()
new ACartesianPoint(
25, 50)
Point1
Point2
ACartesianPoint
new ACartesianPoint(
125, 150)
59
getRadius()
mid(25,50
125,150);
O-O programming with
class methods and
variables
mid(point1x,point1y,
point2x,point2y);

# General Class Structure

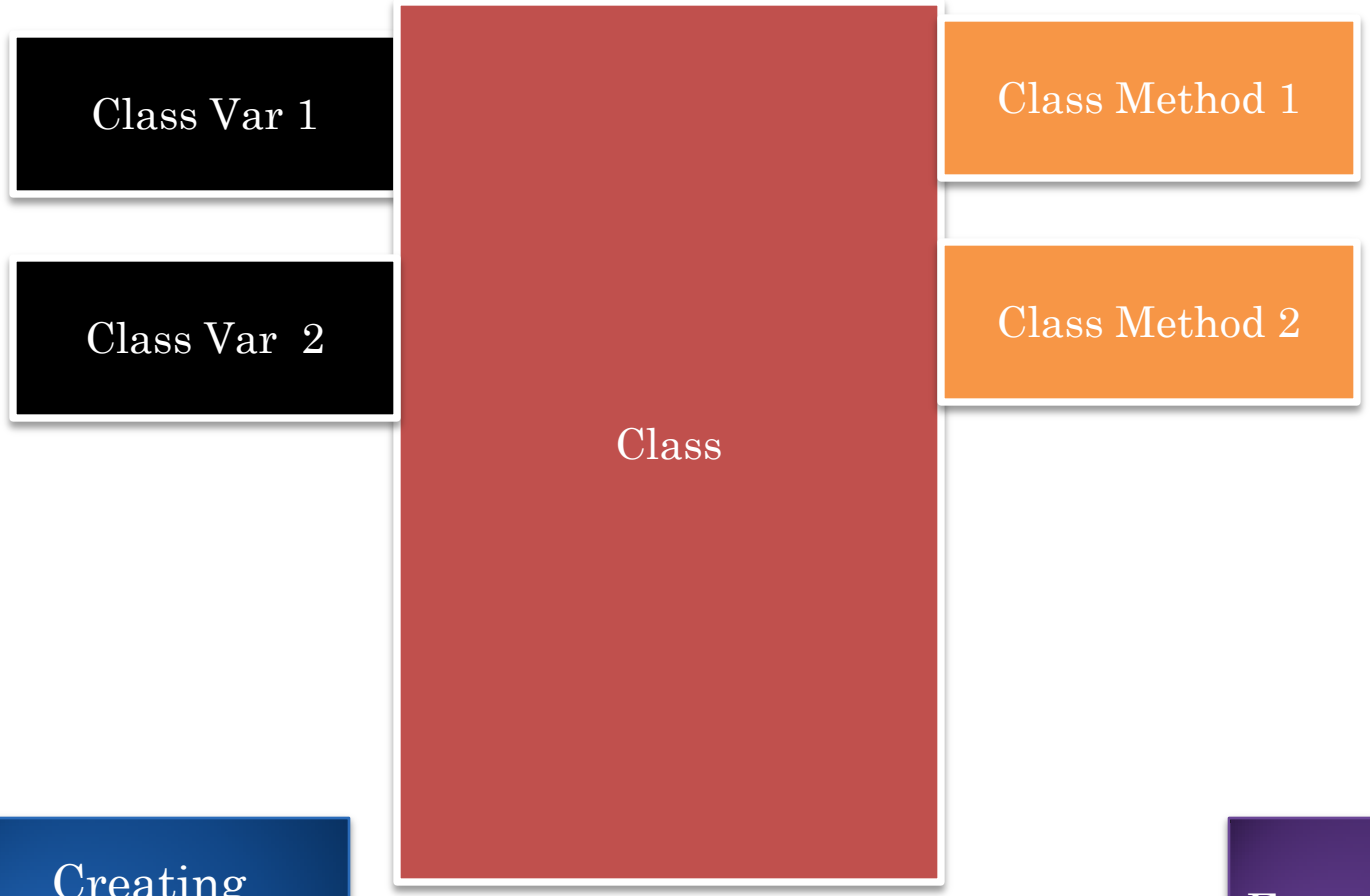


# Example of Class With Instance and Class Methods




Example of Java Class?

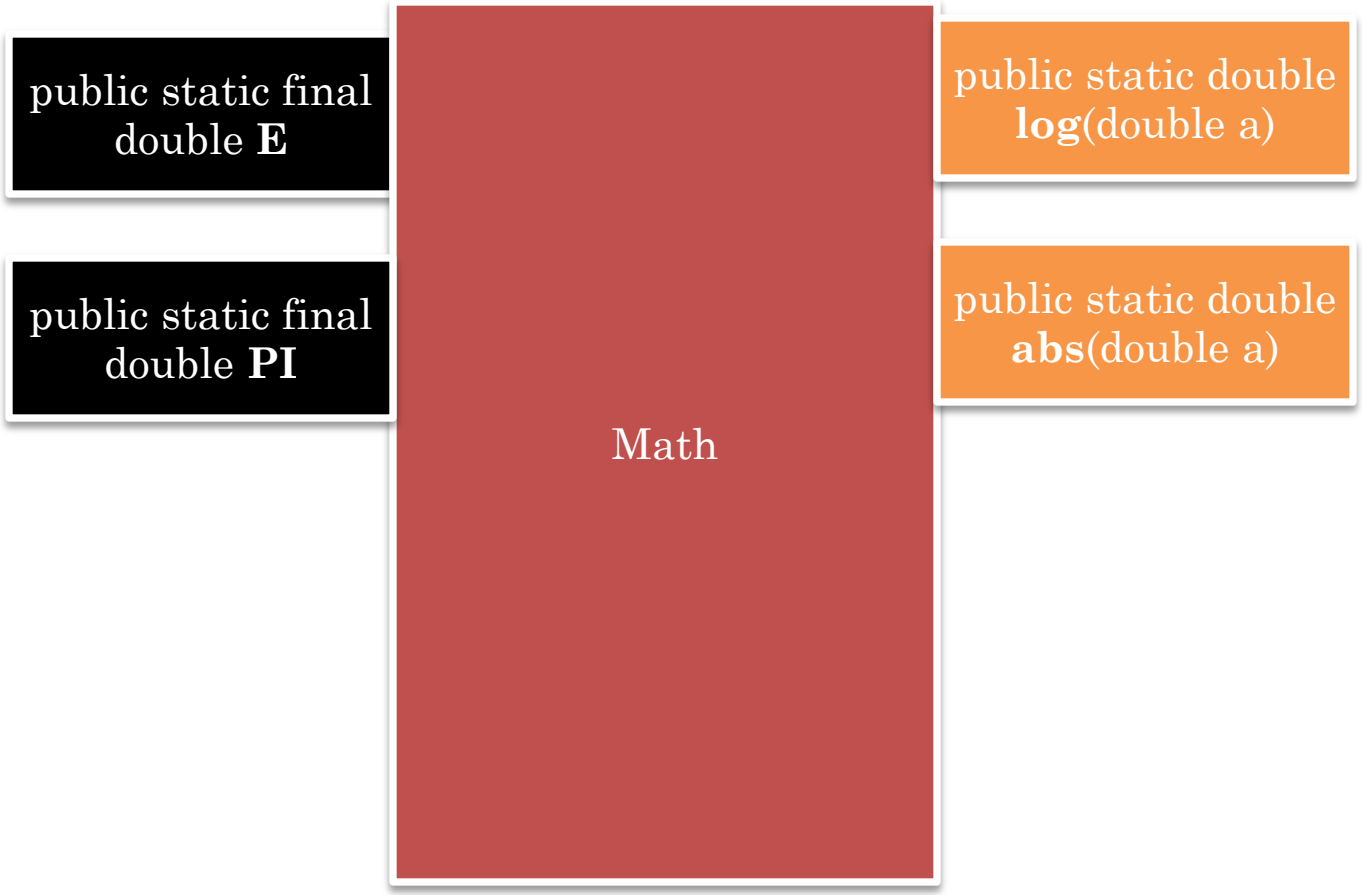
# Class with no Instance Members?



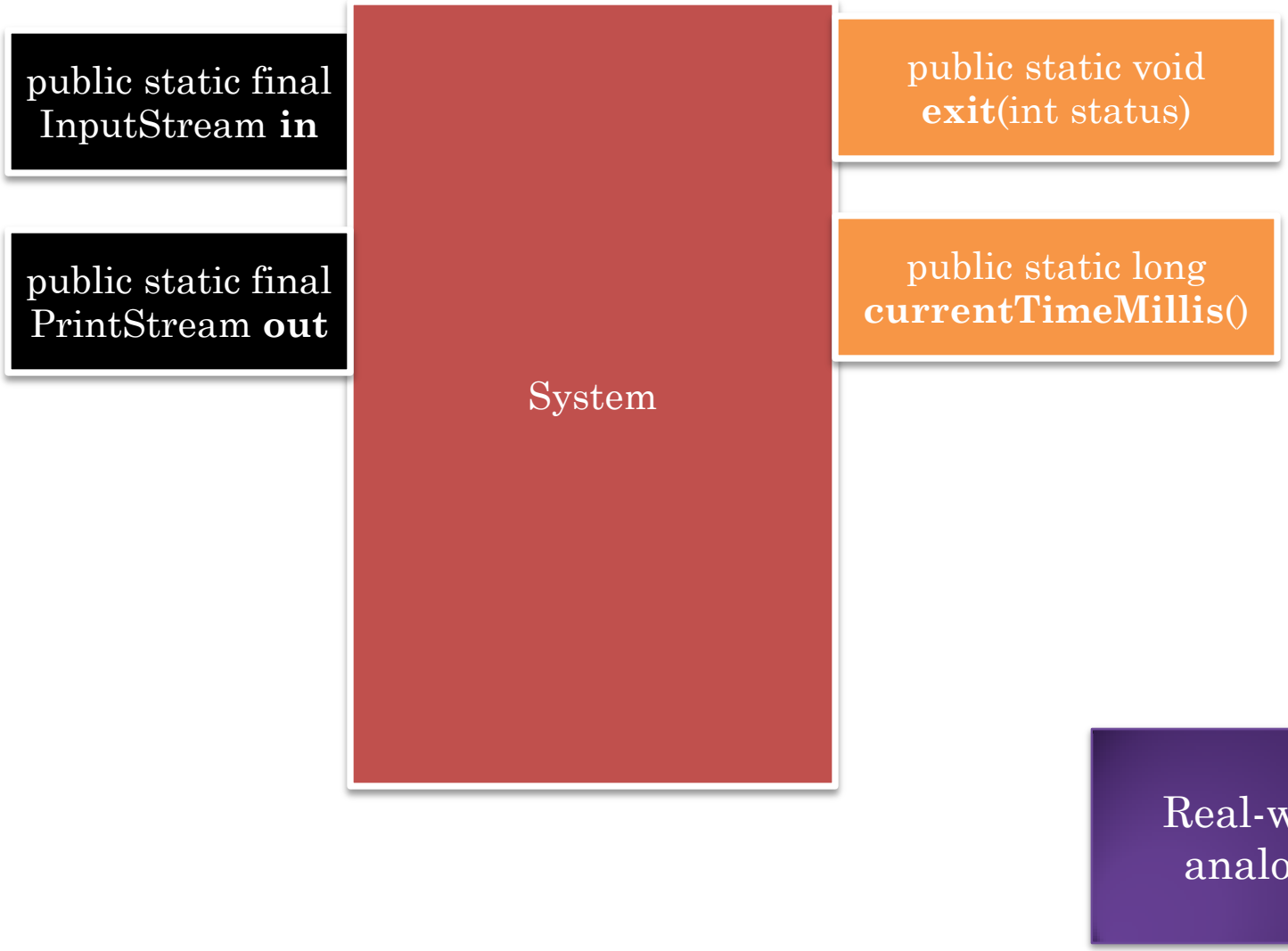

Creating instances meaningless

Example of Java class?

# Math


public static final double **E**
public static final double **PI**
Math
public static double **log**(double a)
public static double **abs**(double a)

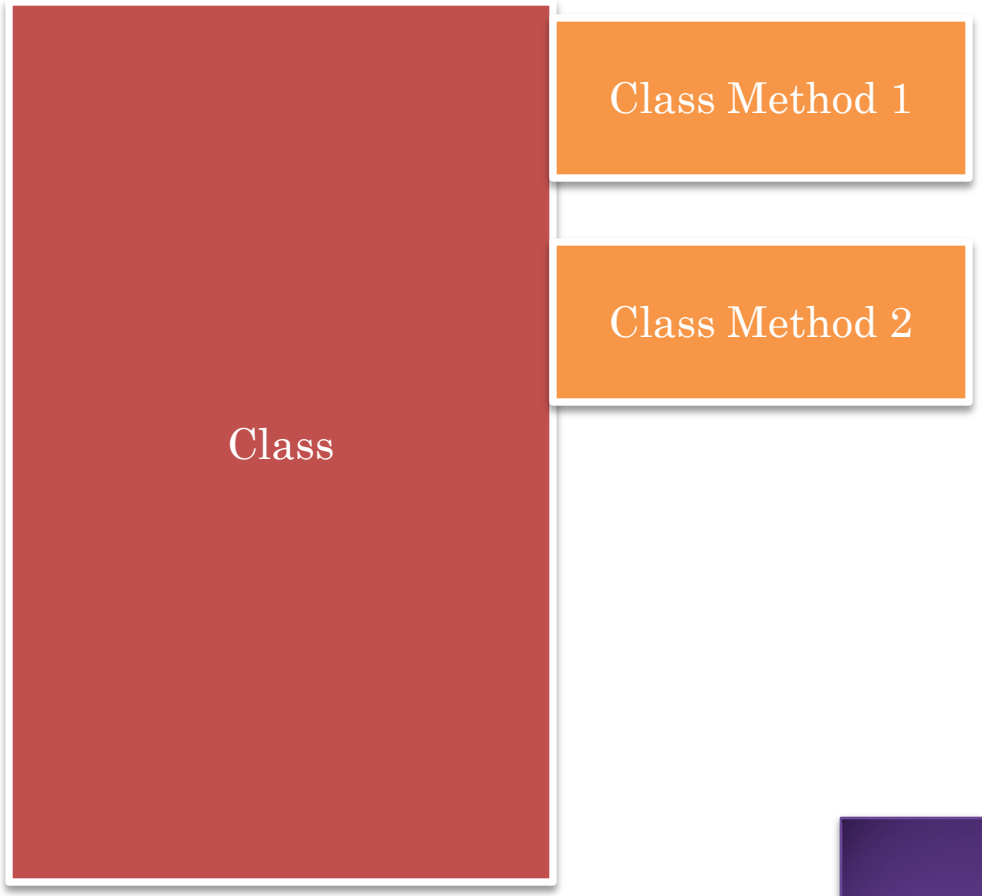
# System


public static final InputStream **in**
public static final PrintStream **out**
System
public static void **exit**(int status)
public static long **currentTimeMillis**()
Real-world analogy?

# Real Life Analogy

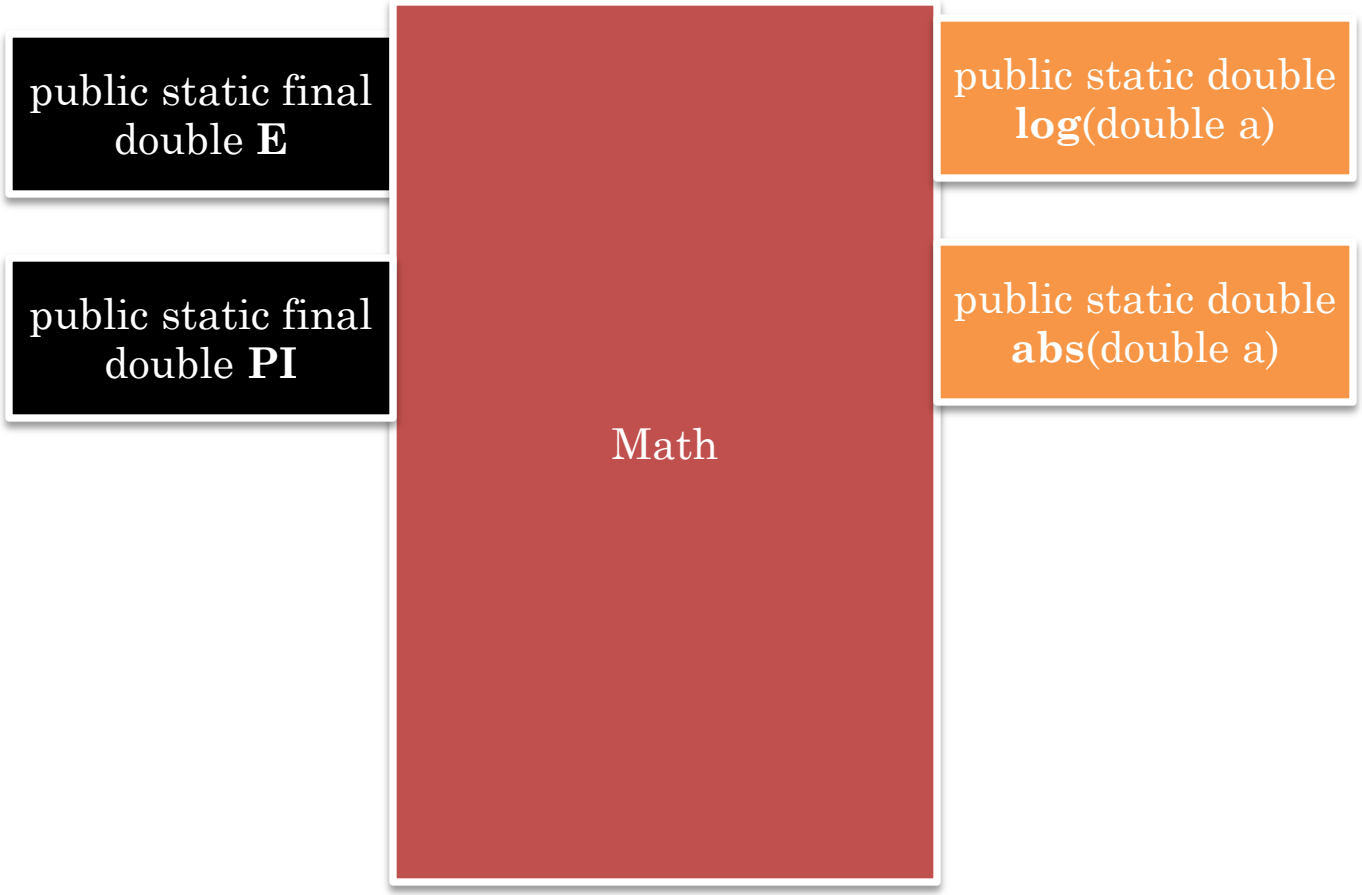

Warehouse, dealership

# Class with Only Class Methods?

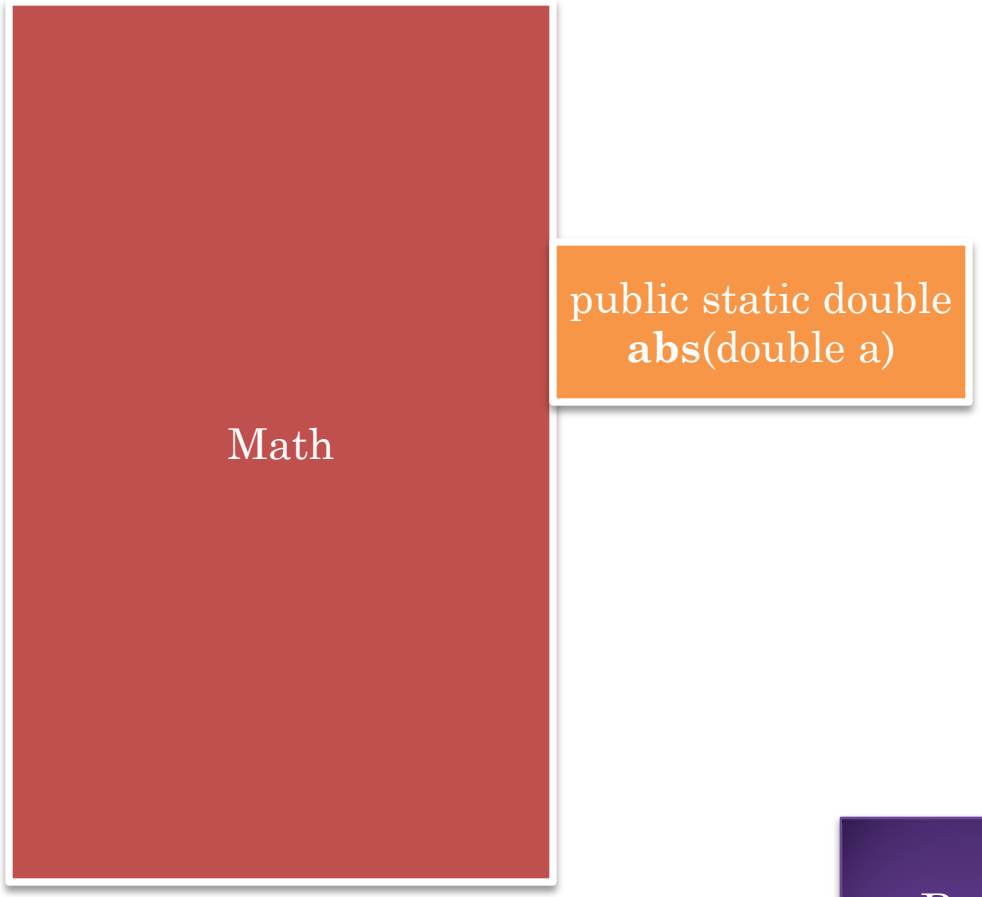



Example?

# Math


public static final double **E**
public static final double **PI**
Math
public static double **log**(double a)
public static double **abs**(double a)

# Modified Math

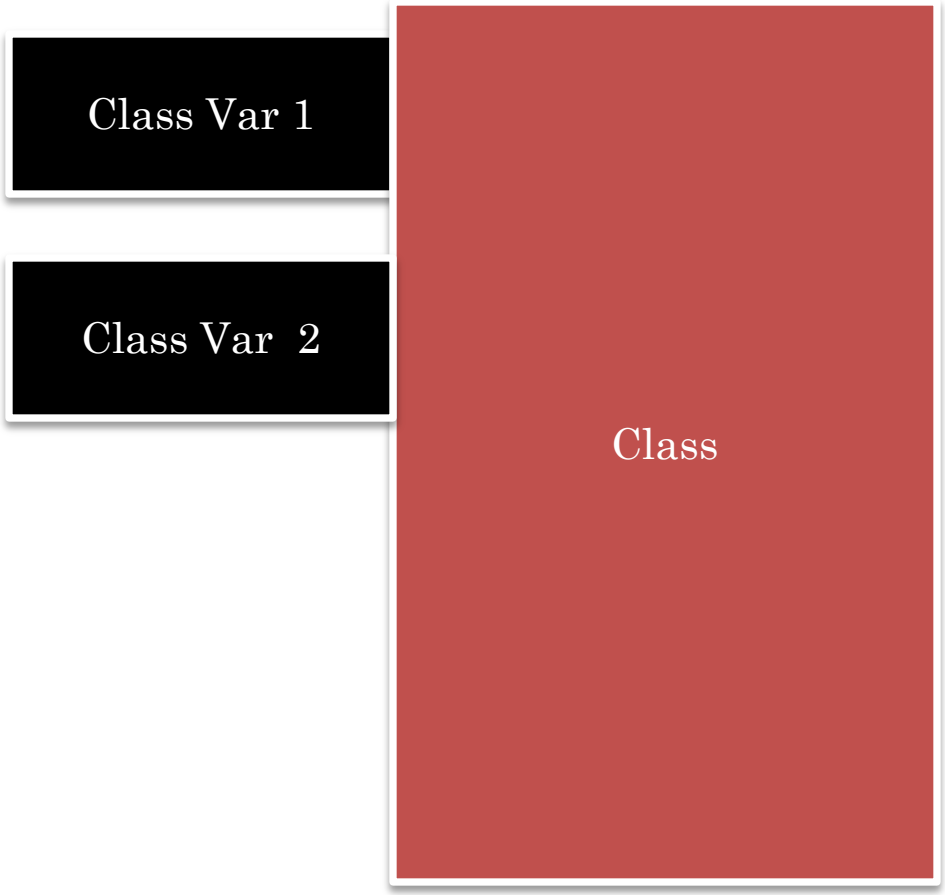



Real-world analogy?

# Real Life Analogy: Pure Service

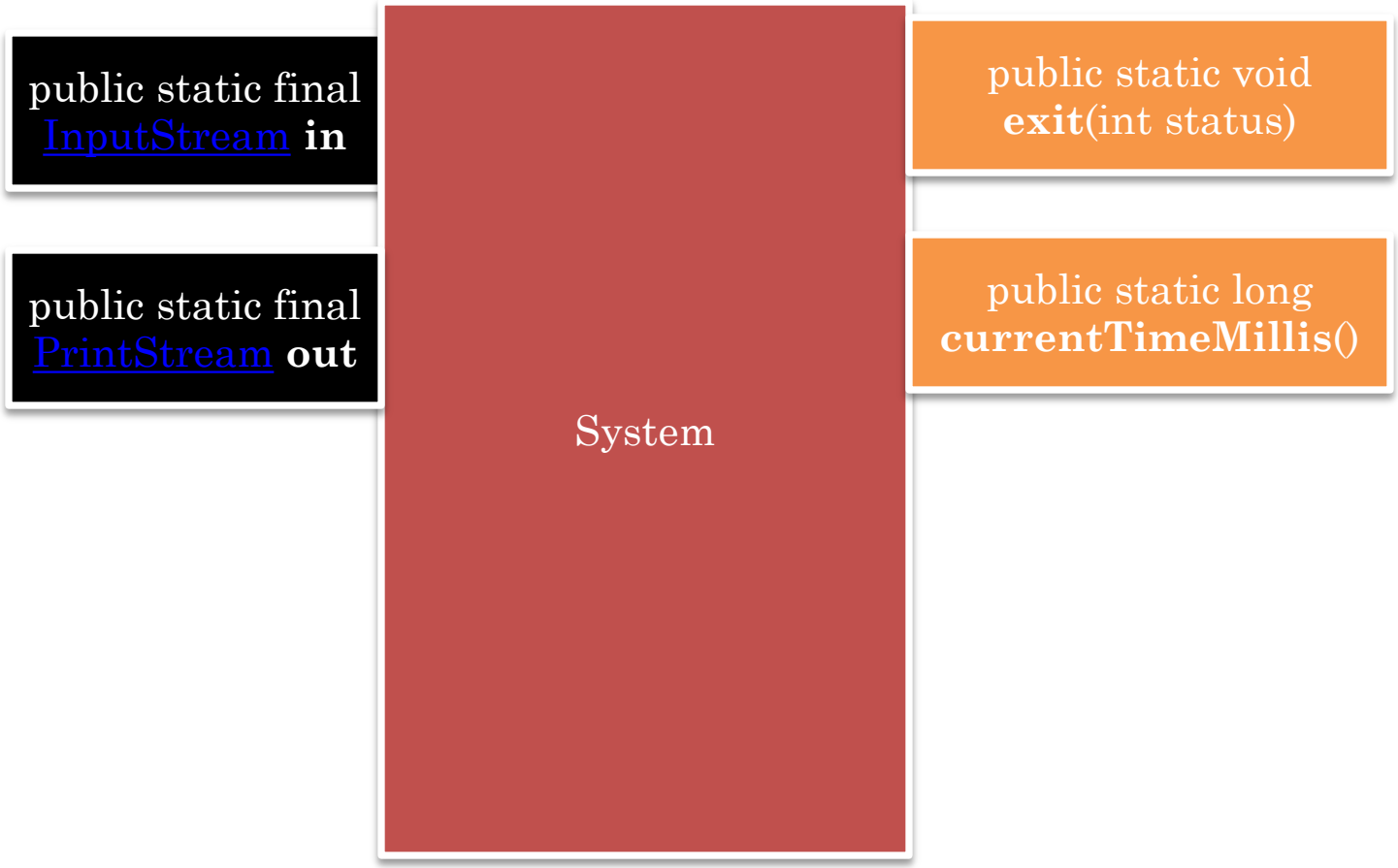

Assume nothing stored

# Class with only Class Variables?

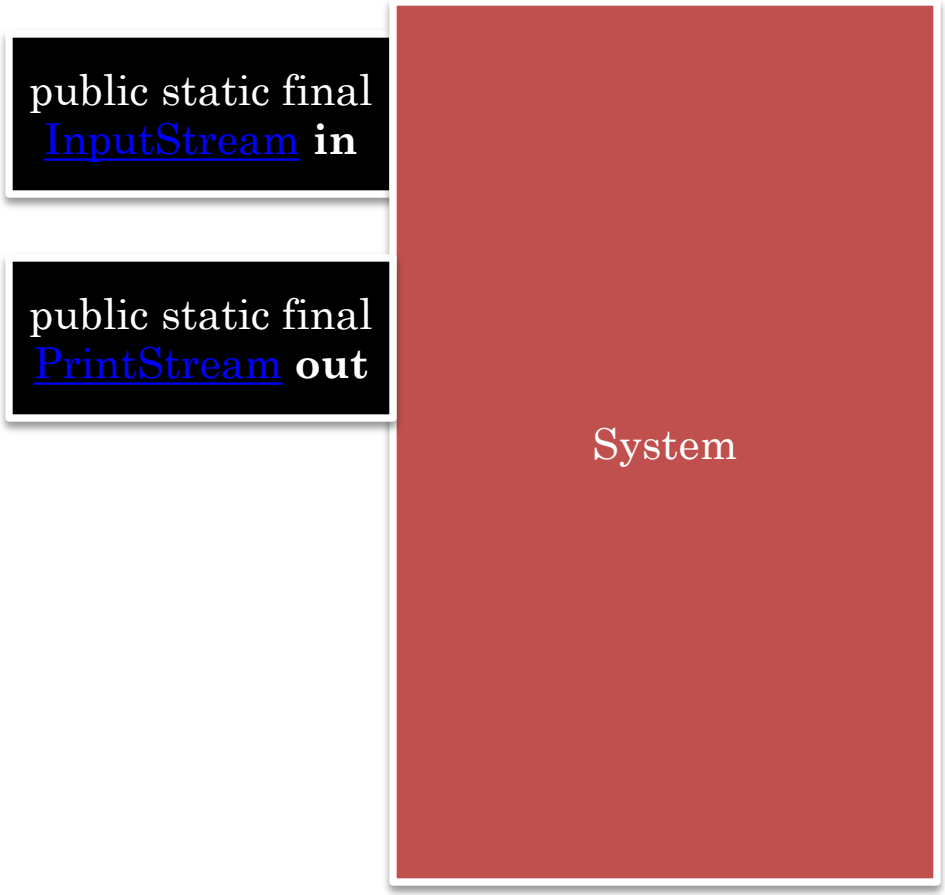



Example?

# System


public static final InputStream in
public static final PrintStream out
System
public static void exit(int status)
public static long currentTimeMillis()

# Modified System




Real-world analogy?

# Unguarded Air Hose

# Class vs. Instance Object

Should some state and associated methods be accessed as a class or instance object

As class object

Multiple independent instances of that state will not occur

Do not need dynamic dispatch or interfaces to that state

E.g: System.in

Singleton class

A class with only one instance, useful when dynamic dispatch or interface needed

# Class or Instance?


Franchise: Multiple franchisees exist, so corresponds to an instance object

# Class or Instance?


One of,  corresponds to a class object

Mom and pop store can later become franchise as it becomes popular and evolves

Class objects often converted to instance objects as program  becomes popular and evolves

When in doubt make it an instance object