SHARE
Technology • Connections • Results

8358
Object-oriented Programming with Java, Part 1

Stephen Pipes
IBM Hursley Park Labs, United Kingdom

Dallas 2003

Objectives

- Explain what objects, classes, methods and messages are
- See how OO is implemented in Java
- Fully understand hello world program!

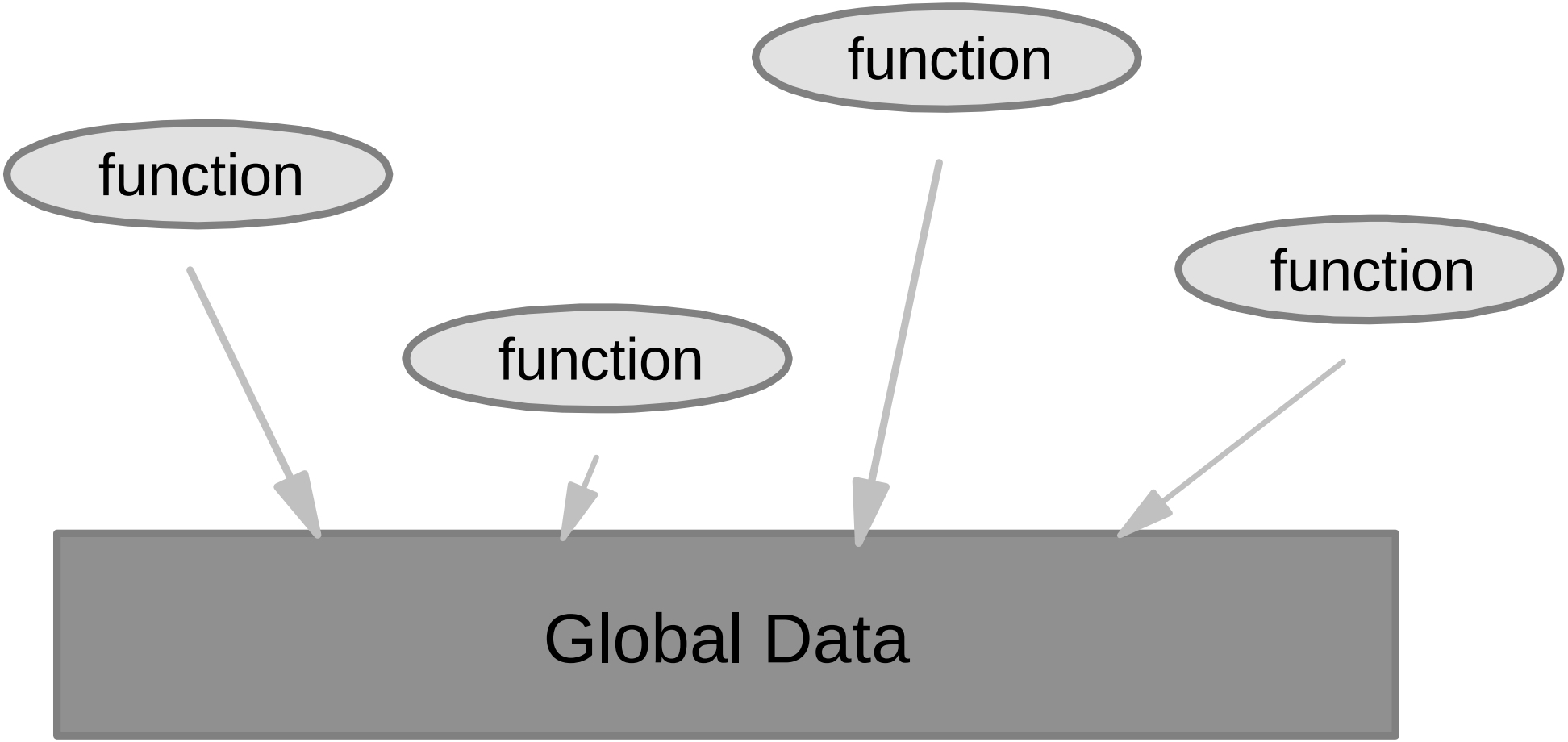
Agenda

- Introduction to classes and objects
- The lifecycle of an object
- Public data members
- Class data and methods
- Hello World!

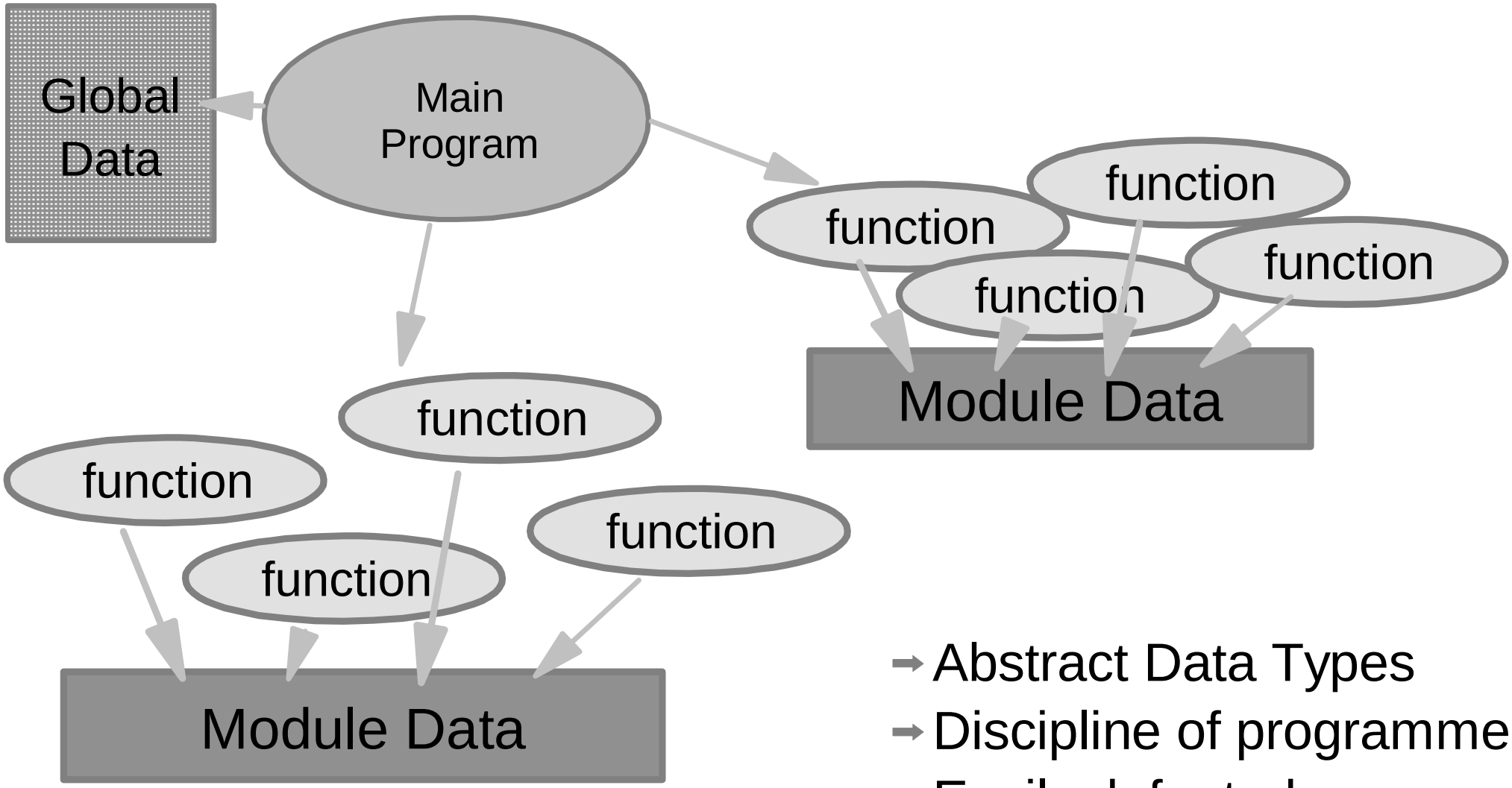
Hello World


```
/**
 * The HelloWorldApp class implements an application that
 * simply displays "Hello World!" to the standard output.
 */
class HelloWorldApp {
    public static void main(String[] args) {
        System.out.println("Hello World!"); //Display the string.
    }
}
```

Procedural Programming



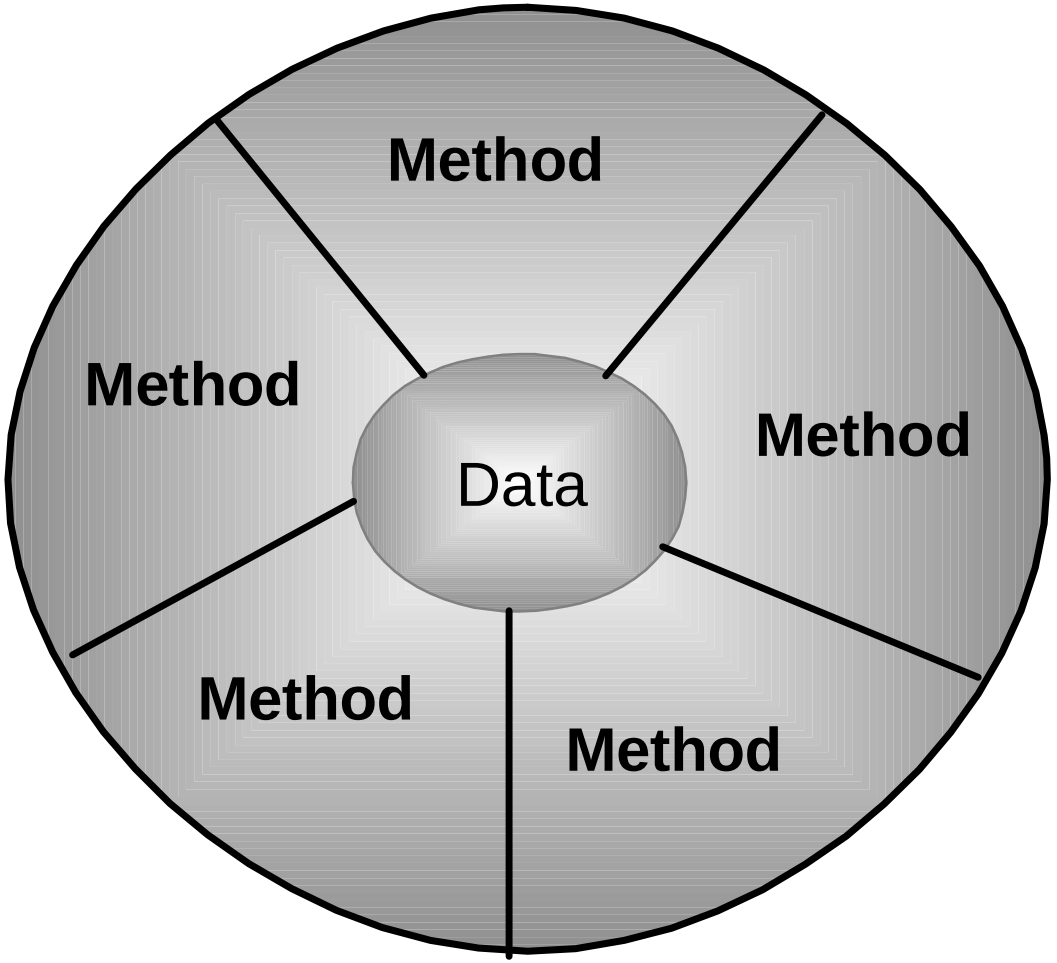
Modularization



- Abstract Data Types
- Discipline of programmer
- Easily defeated

Classes

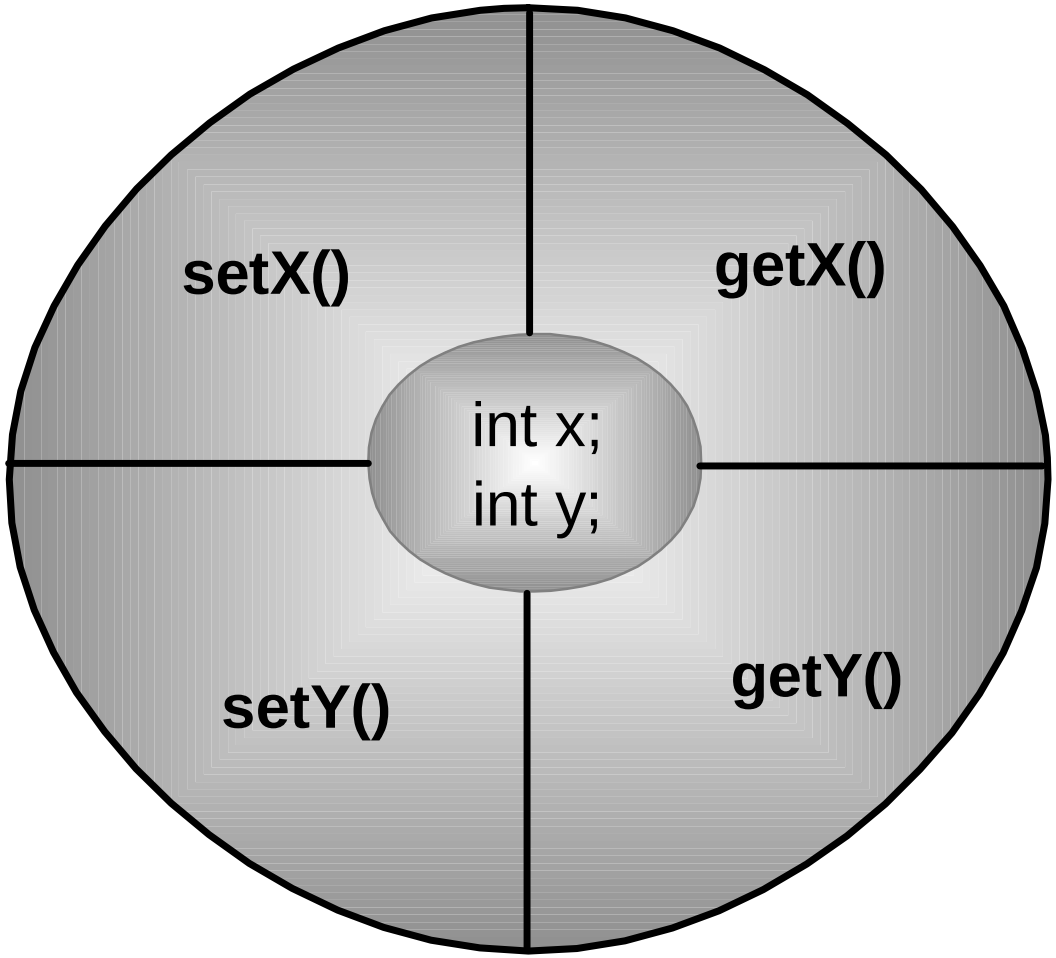
- Encapsulation
- Data Structures
- Methods (behaviour)
- Like defining a new type



A Class in Java

```
class SimplePoint {

  // Data encapsulated by the class
  private int x;
  private int y;

  // Methods that form external interface
  public int getX() { return x; }
  public int getY() { return y; }
  public void setX(int pos) { x = pos; }
  public void setY(int pos) { y = pos; }
}
```

SimplePoint



"Finding" Classes

- Look for nouns in the problem analysis

"Customers phone in and place an order for one or more items. The customer service representative creates a new order and adds the items to it. Next the shipping address and payment details are taken so that the order can be shipped and the customer's account charged".

- Customer
- Order
- Item
- CustomerServiceRepresentative
- Shipping Address
- PaymentDetails
- Account

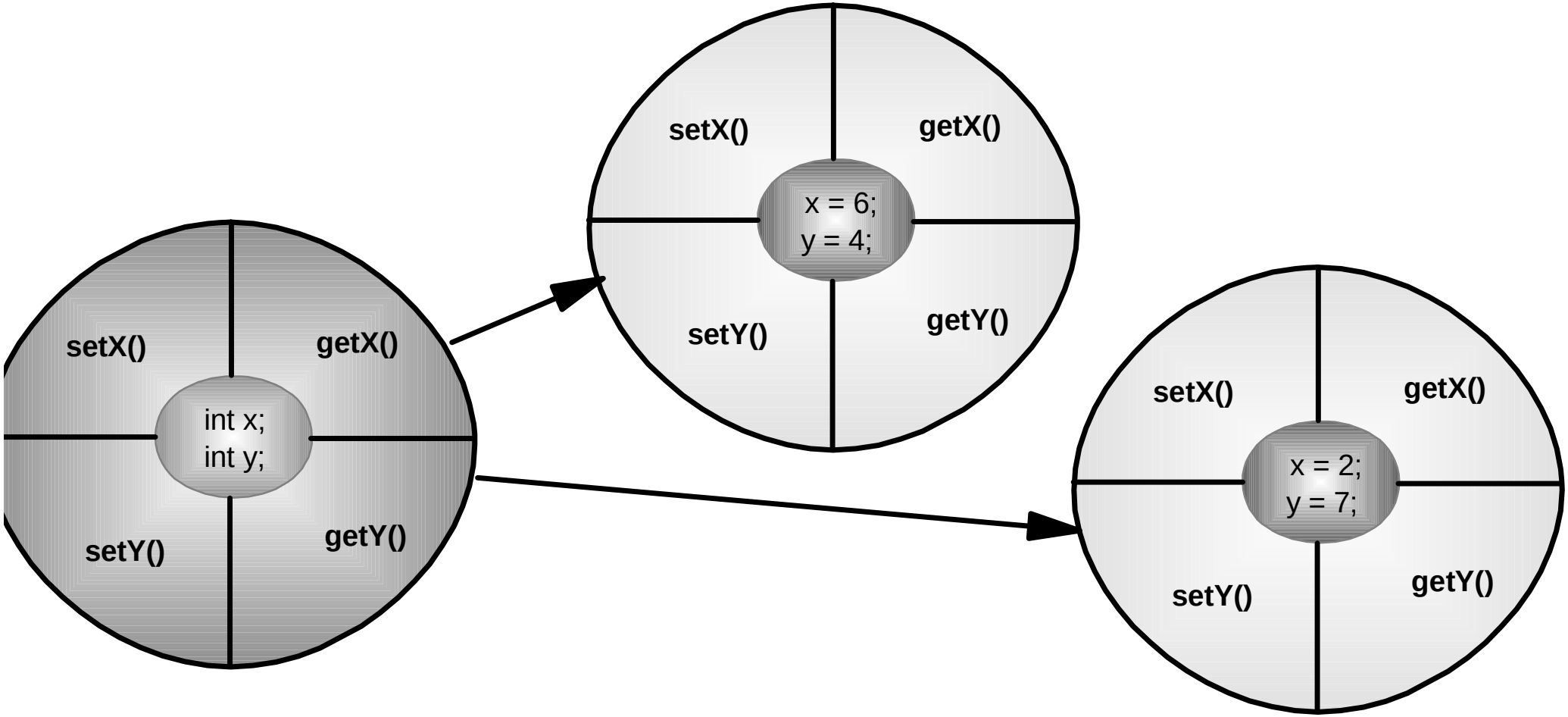
"Finding" Methods

- Look for verbs in the problem analysis
- Identify the class that has the major responsibility for carrying it out

"Customers phone in and place an order for one or more items. The customer service representative creates a new order and adds the items to it. Next the shipping address and payment details are taken so that the order can be shipped and the customer's account charged".

Objects

- If a class is like a new type,
- Then an object is a variable of that type
  - Called an "instance" of the class



Agenda

- Introduction to classes and objects
- The lifecycle of an object
- Public data members
- Class data and methods
- Hello World!

The Lifecycle of an Object

- Creating Objects
- Using Objects
- Cleaning up unused Objects

Objects in Java


```
{
    SimplePoint aPoint;
    SimplePoint anotherPoint;

    aPoint = new SimplePoint();
    aPoint.setX(6);
    aPoint.setY(4);

    anotherPoint = new SimplePoint();
    anotherPoint.setX(2);
    anotherPoint.setY(7);
```

Creating Objects

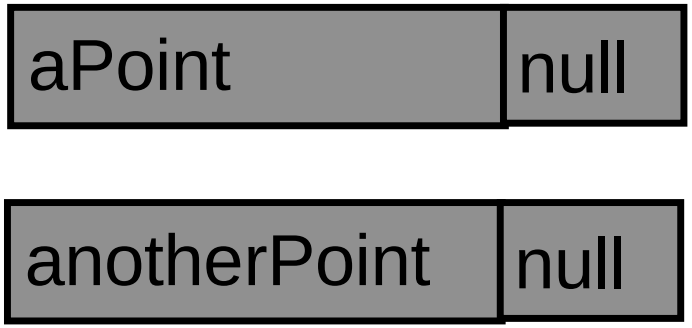
- SimplePoint aPoint;

Creating Objects

SHARE
Technology · Connections · Results

SimplePoint aPoint;

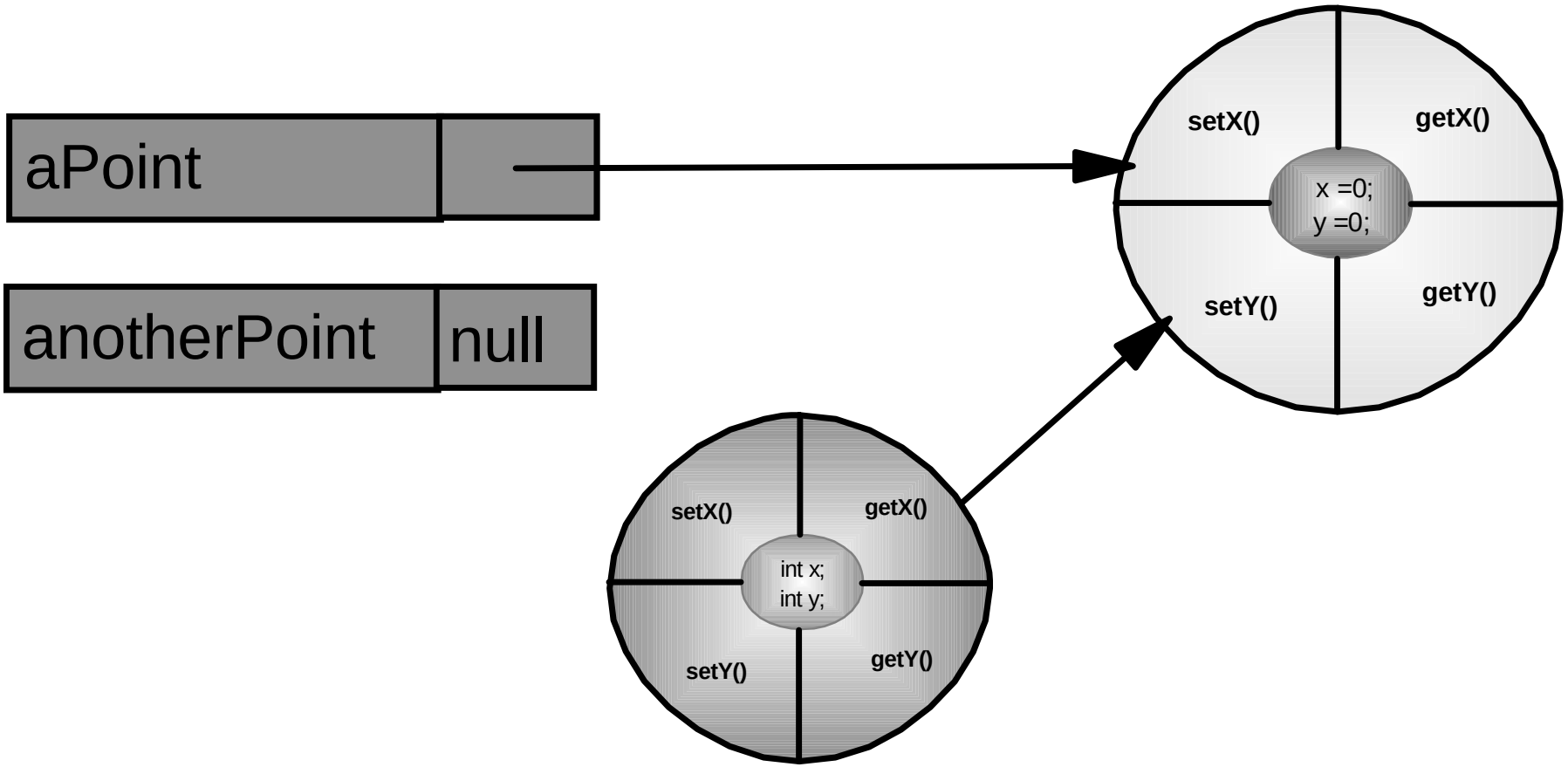
➡ SimplePoint anotherPoint;



Creating Objects

SimplePoint anotherPoint;

➡ aPoint = new SimplePoint();



Creating Objects

- The new SimplePoint object is an *instance* of the SimplePoint class.
- Creating an instance of a class is called *instantiation*.
- How does the class make a new instance of itself?

Constructors

aPoint = new SimplePoint();

- The new operator allocates memory for a new object of the appropriate type
- new requires a single postfix operator, a call to a *constructor*
- A constructor is a special method defined on a class that initialises the state of an object of that class when it is *instantiated*

Constructors

- Every class has a default constructor that takes no arguments
- You can provide as many constructors as you like, as long as they are differentiated by the number and type of their arguments
  - Have many methods of the same name but with differing arguments is called *overloading* or *polymorphism*

Give me an example, *please*

```
class SimplePoint {

  // Data encapsulated by the class
  private int x;
  private int y;

  // Methods that form external interface
  public int getX() { return x; }
  public int getY() { return y; }
  public void setX(int pos) { x = pos; }
  public void setY(int pos) { y = pos; }
}
```

- **Default constructor: the method you can't see!**

```
public SimplePoint() { }
```

A constructor method has the same name as the class, and no return type.

More constructor examples

```
class SimplePoint {

  // Data encapsulated by the class
  private int x;
  private int y;

  // Constructors
  public SimplePoint(int xpos, int ypos) {
    x = xpos; y = ypos;
  }

  public SimplePoint(int radius, double theta) {
    x = (int) (radius * Math.cos(theta));
    y = (int) (radius * Math.sin(theta));
  }
...
```

Using the Constructors

- aPoint = **new** SimplePoint();
  - creates new SimplePoint object and then calls default constructor method
- aPoint = **new** SimplePoint(3,5);
  - creates new SimplePoint object and then calls the (int xpos, int ypos) constructor method
- aPoint = **new** SimplePoint(3,15.5d);
  - creates new SimplePoint object and then calls the (int radius, double theta) constructor method

The Lifecycle of an Object

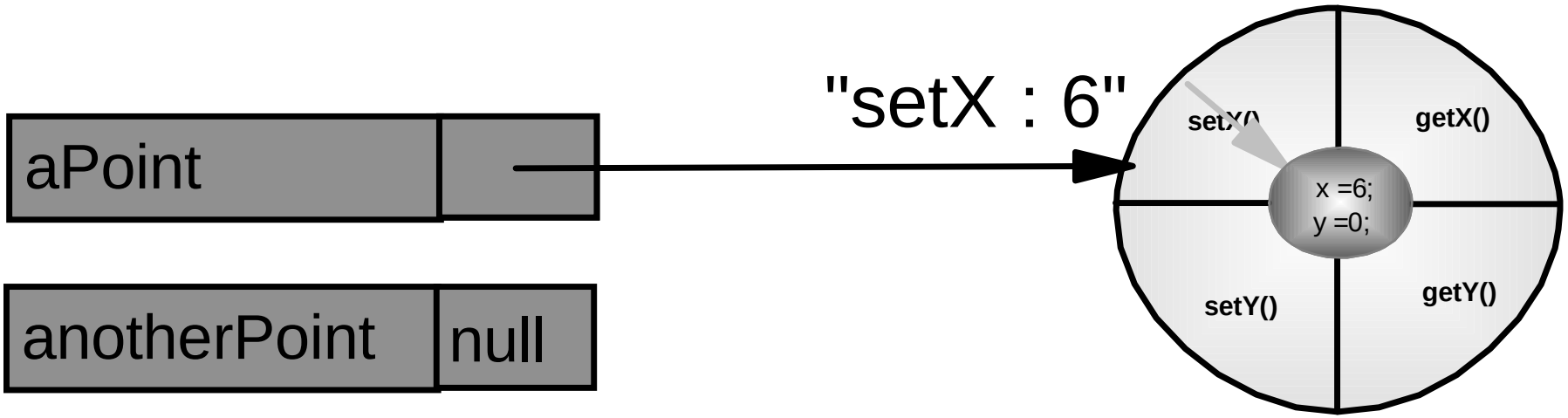
- Creating Objects

  Using Objects
- Cleaning up unused Objects

Using Objects

aPoint = new SimplePoint();

➡ aPoint.setX(6);



Messages and Methods

- We say that the "setX" *message* has been sent to the aPoint object.
- On receiving the setX message, the aPoint object invokes the setX method
- The "dot" or "message" operator "." is used to send a message to an object

Object.message(message parameters)

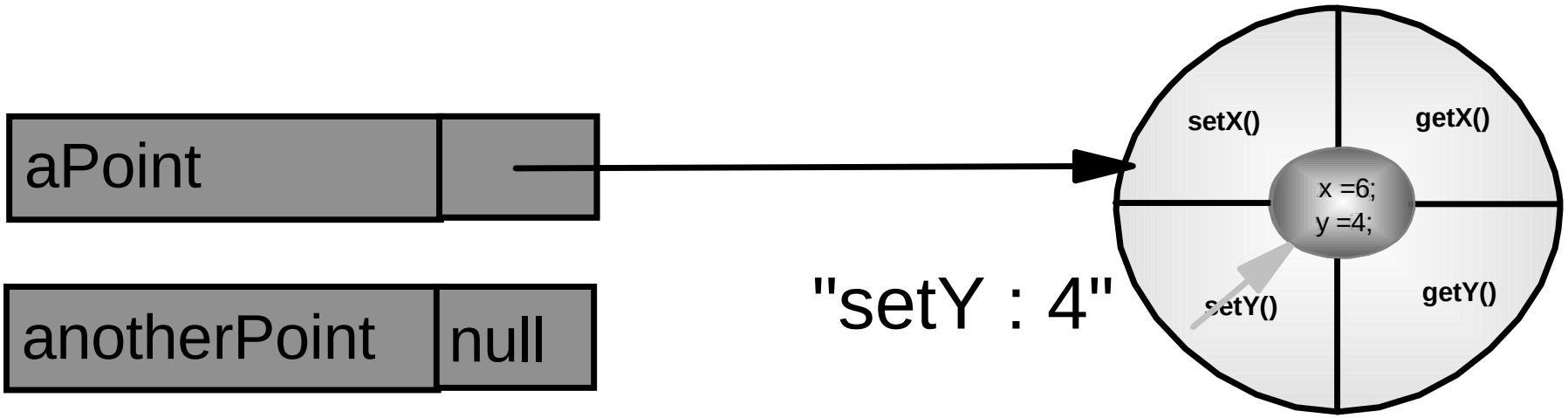
Messages and Methods

- The value of the expression object.message is the return value of the message method
  - aPoint.getX() has the integer value 6
- You can use the dot notation in expressions
- The dot operator associates left to right
  - upperCaseName = obj.getName().toUpperCase();

Using Objects

aPoint.setX(6);

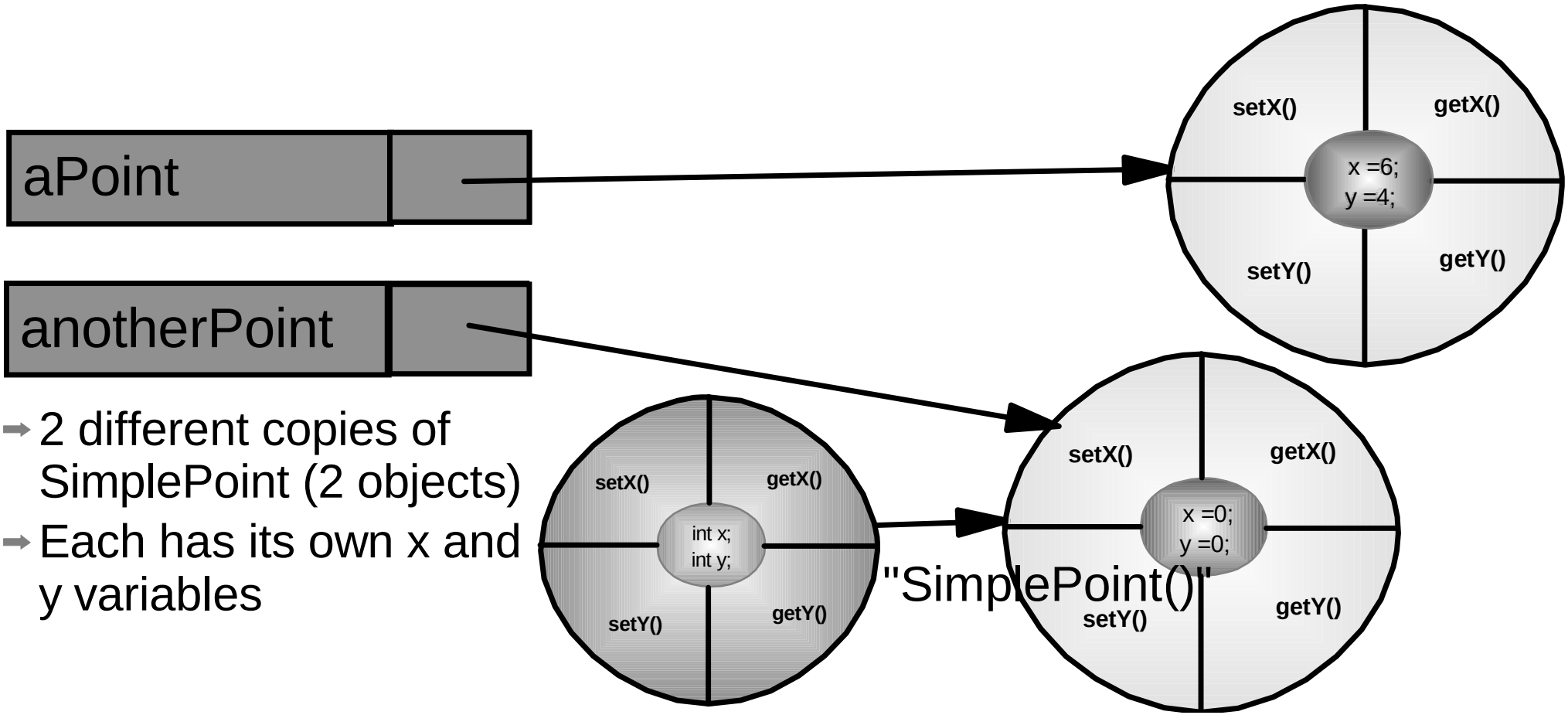
➡ aPoint.setY(4);



Using Objects

aPoint.setY(4);

- anotherPoint = new SimplePoint();

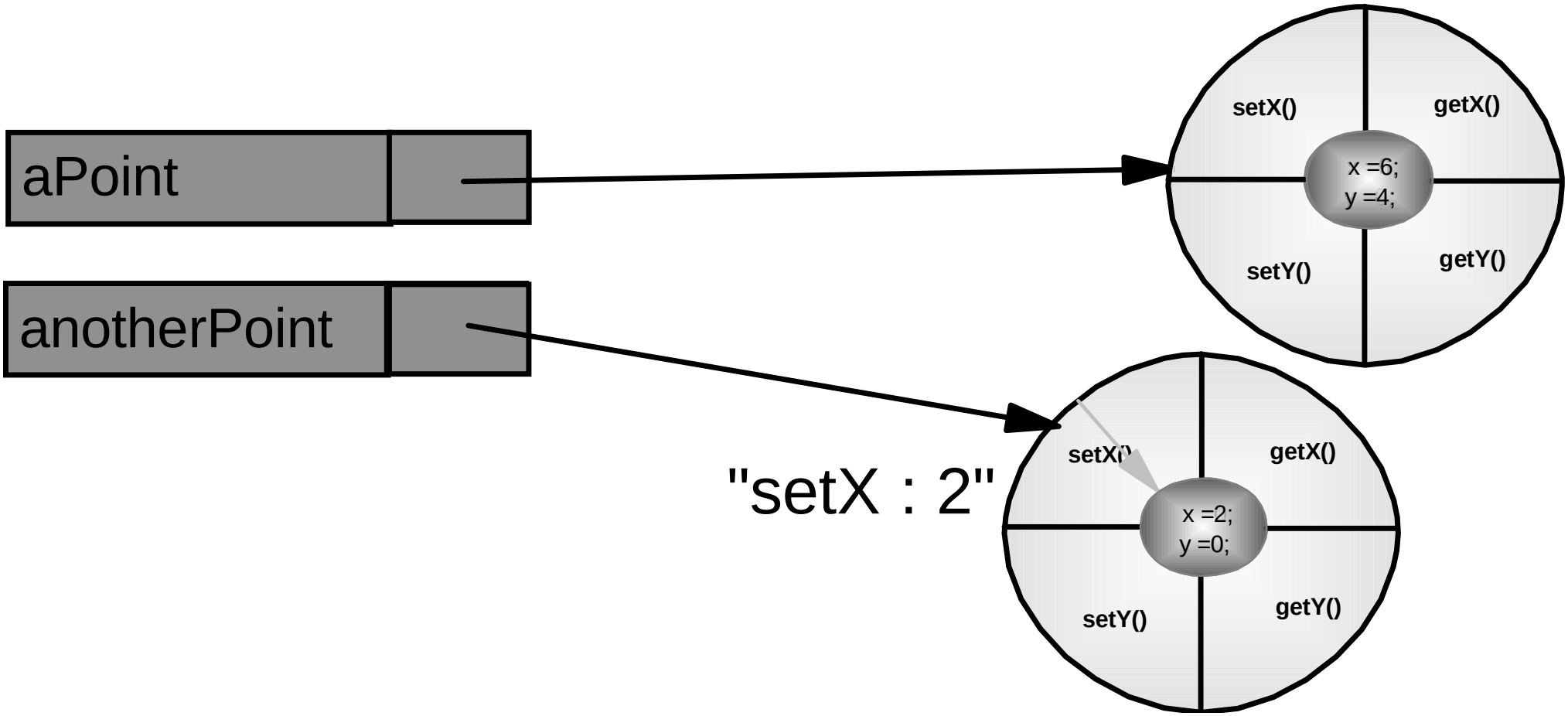


- 2 different copies of SimplePoint (2 objects)
- Each has its own x and y variables

Using Objects

anotherPoint = new SimplePoint();

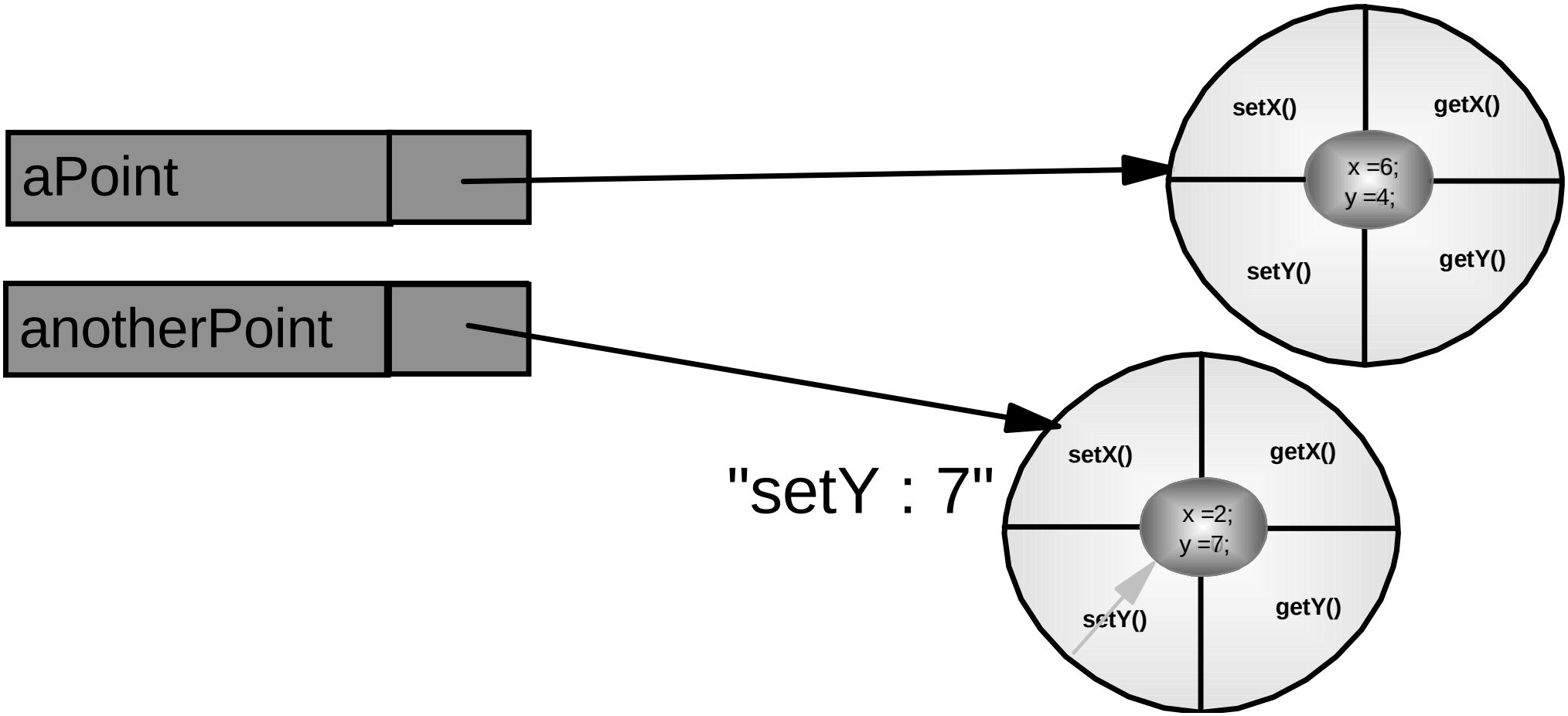
➡ anotherPoint.setX(2);



Using Objects

anotherPoint.setX(2);

➡ anotherPoint.setY(7);



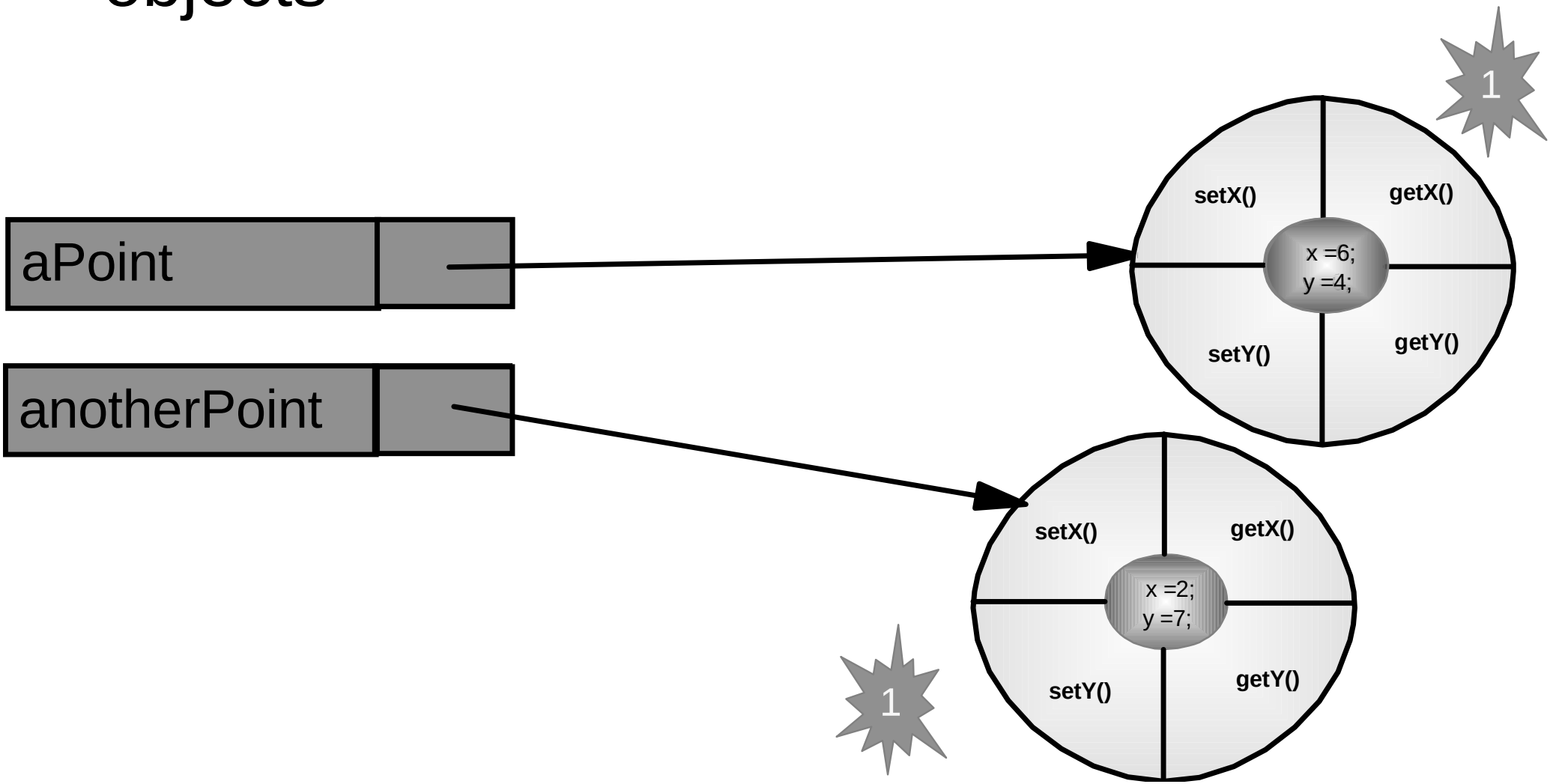
The Lifecycle of an Object

- Creating Objects
- Using Objects

  Cleaning up unused Objects

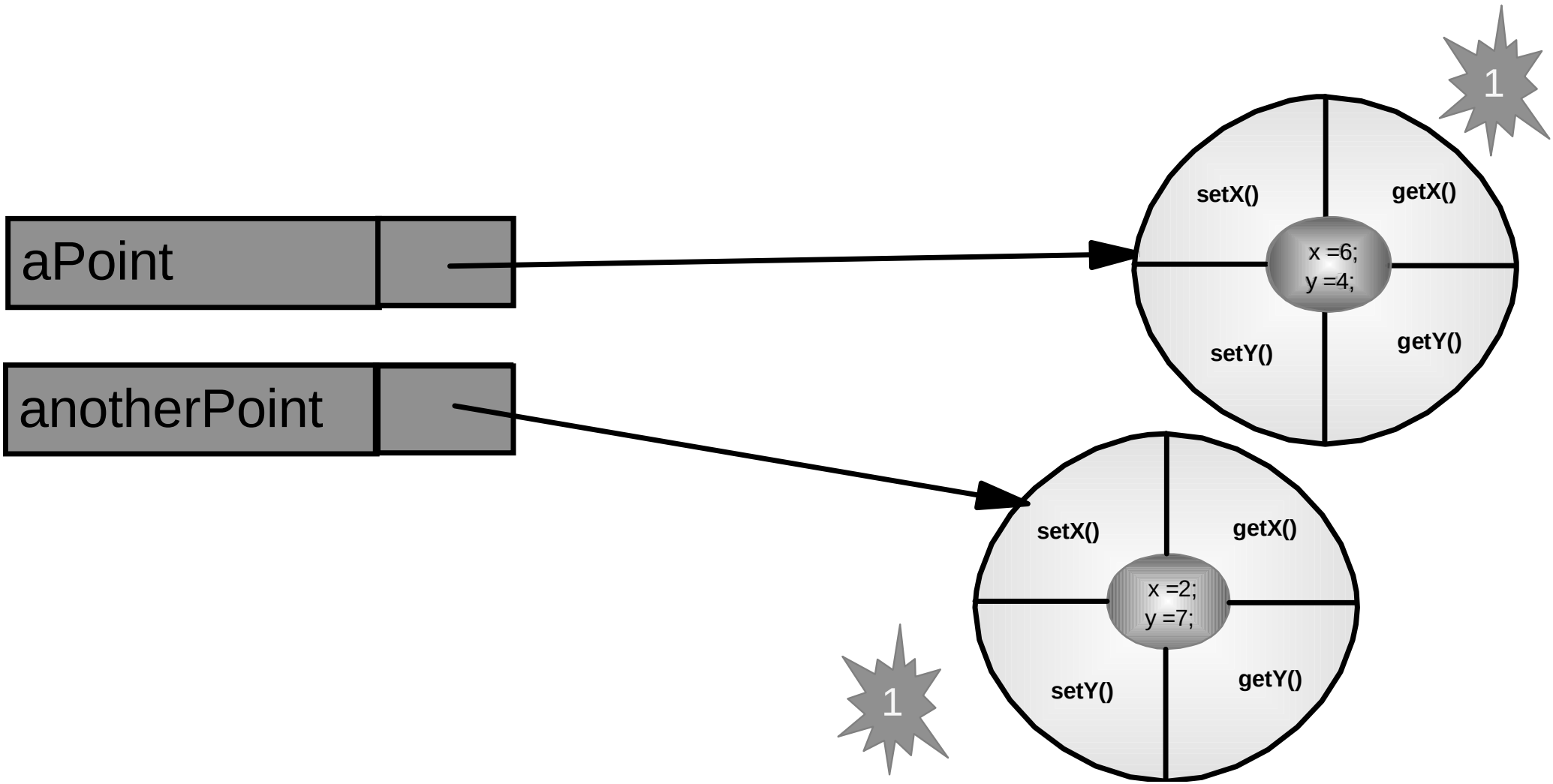
Reference Counting

- Java Runtime keeps track of references to objects



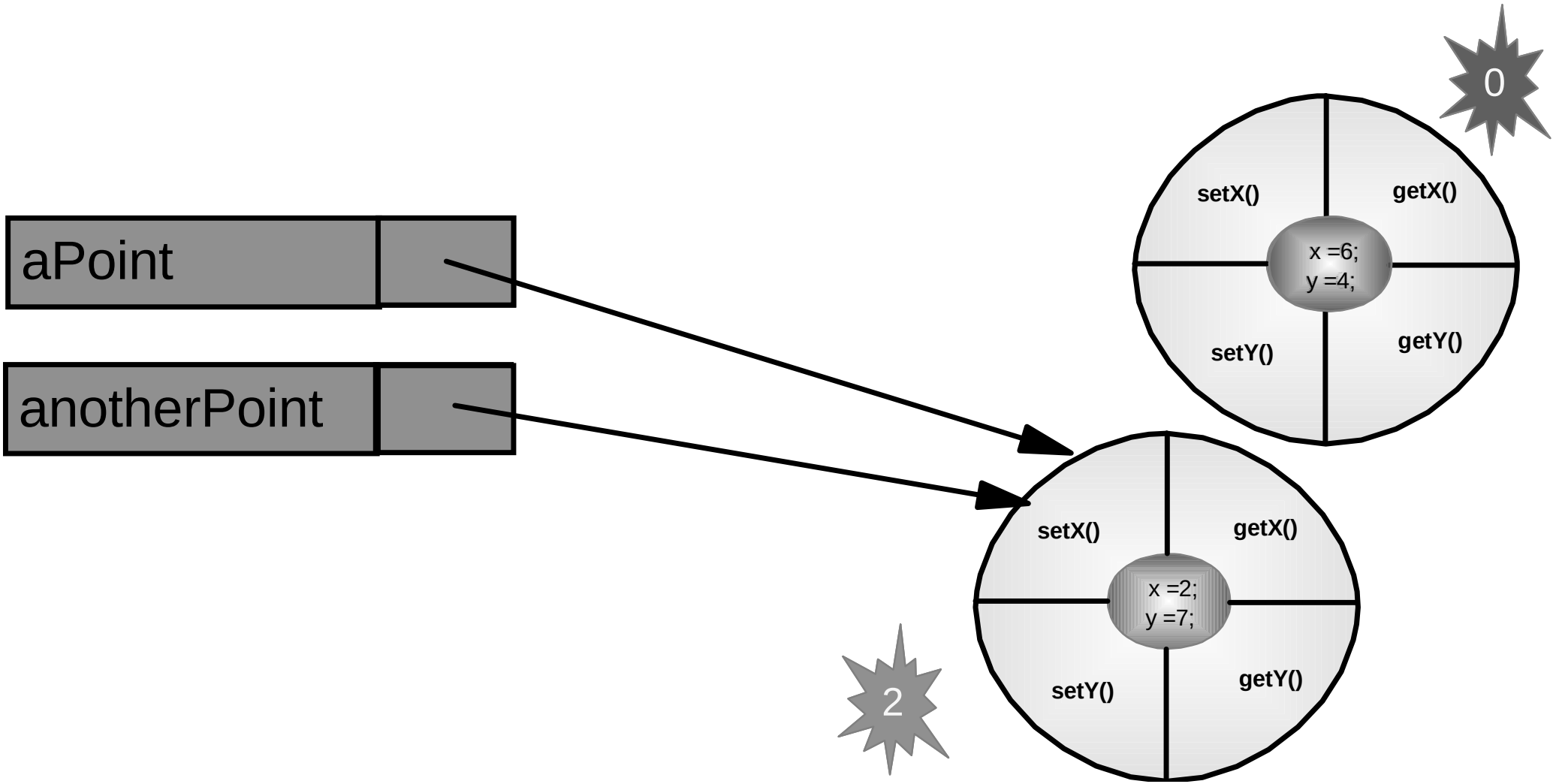
Reference Counting

- aPoint = anotherPoint;



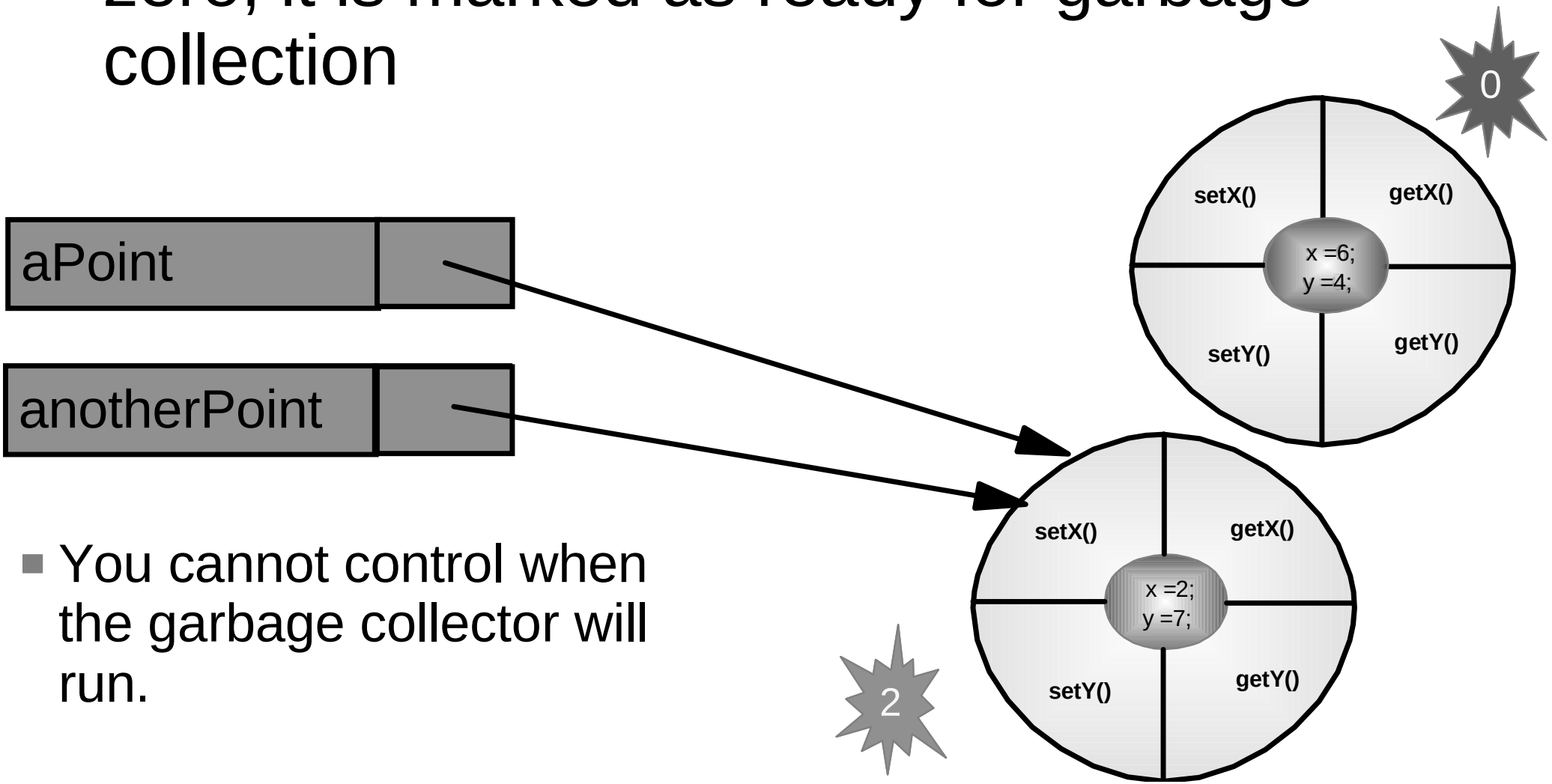
Reference Counting

- aPoint = anotherPoint;



Cleaning up unused objects

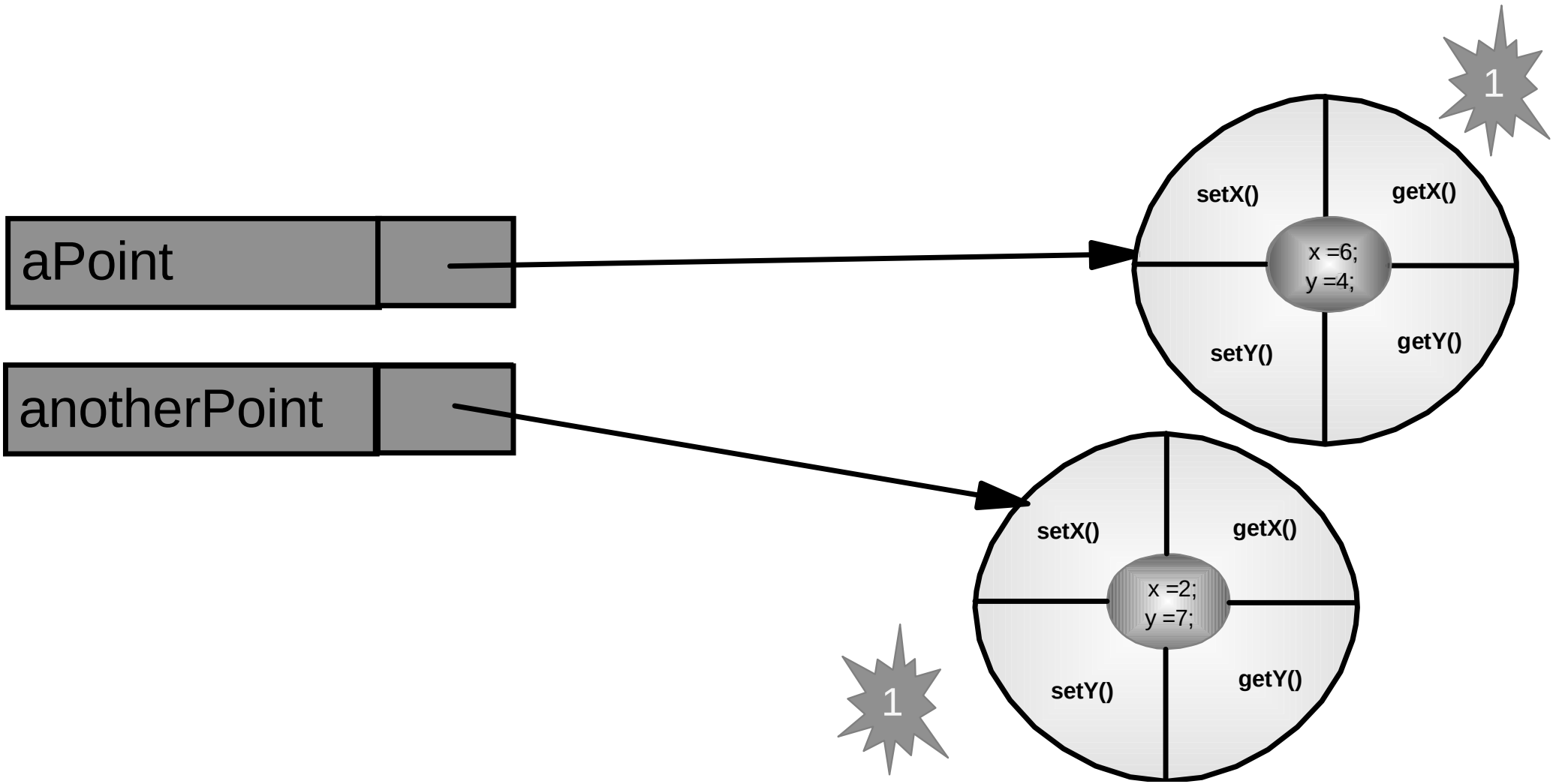
- When an object's reference count drops to zero, it is marked as ready for garbage collection



- You cannot control when the garbage collector will run.

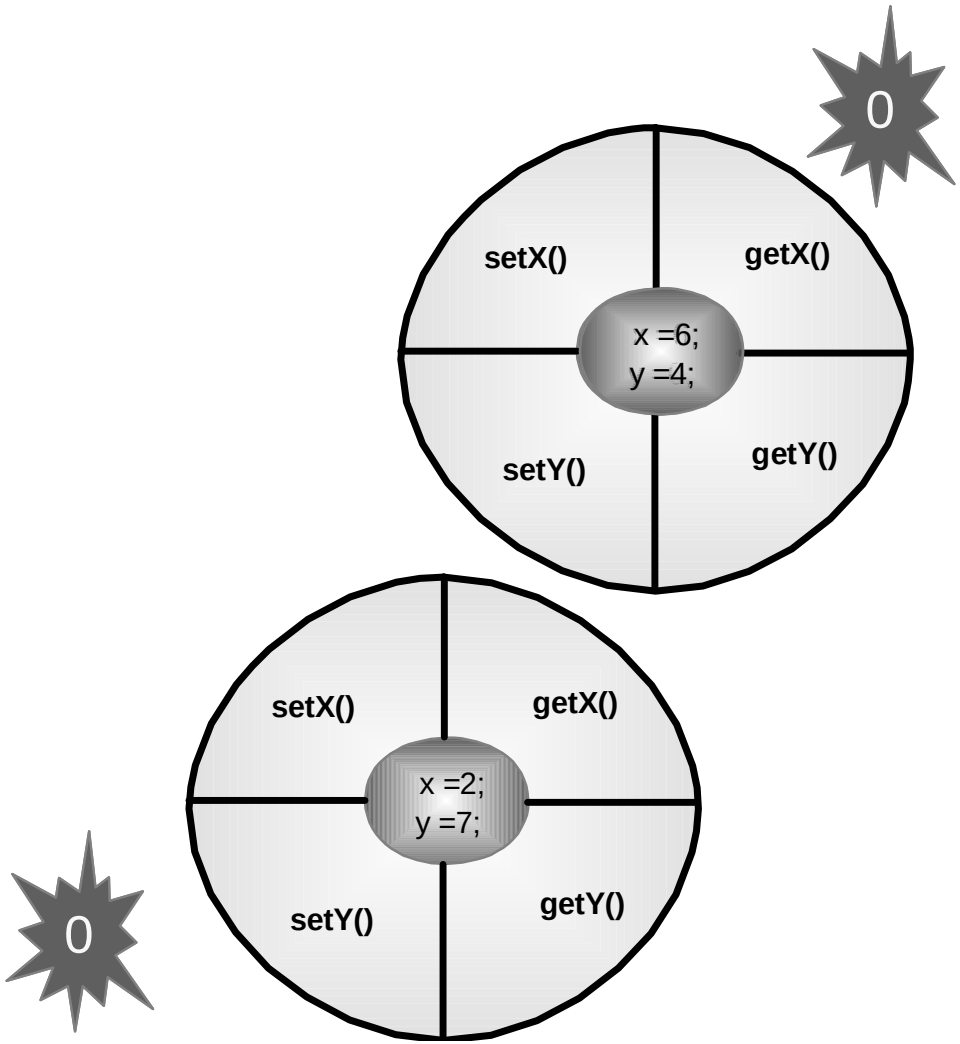
Cleaning up unused objects

- }



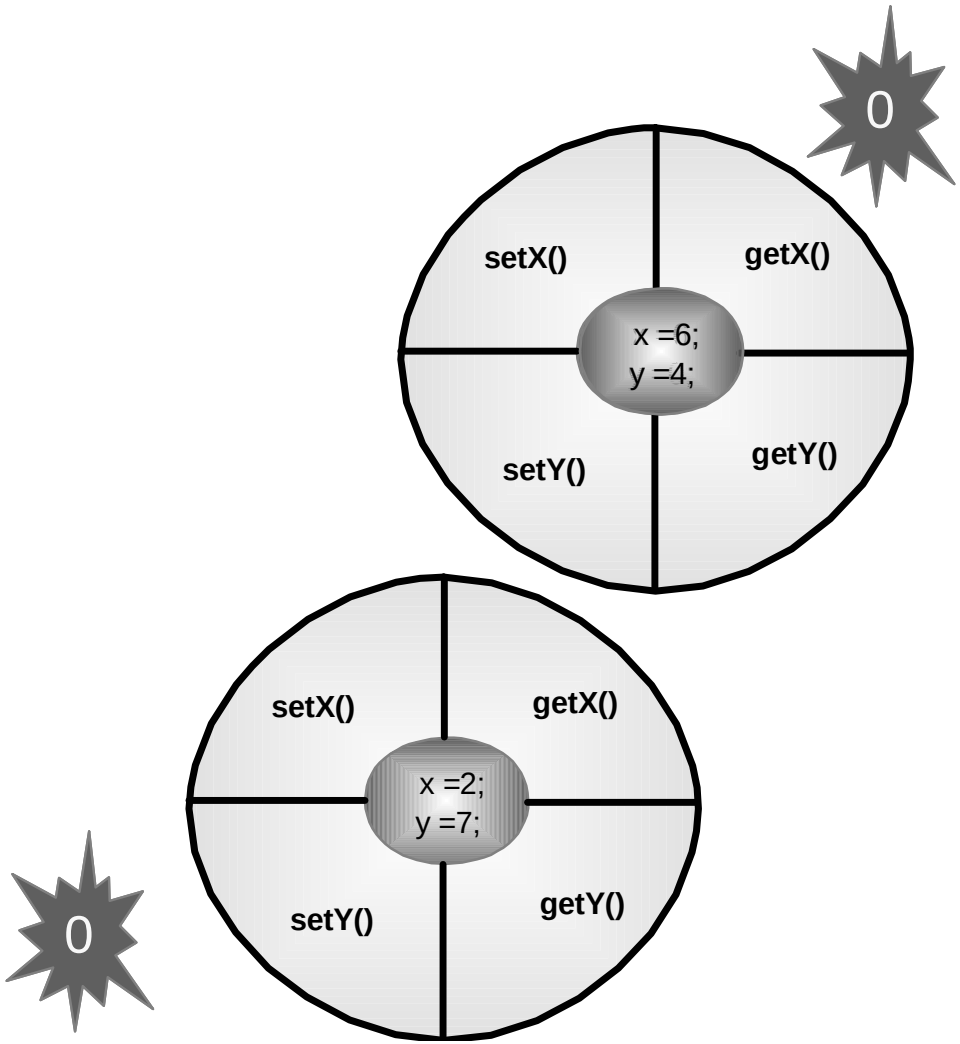
Cleaning up unused objects

- }



Sometime later...

- the garbage collector runs



Sometime later...

- the garbage collector runs

Agenda

- Introduction to classes and objects
- The lifecycle of an object
- Public data members
- Class data and methods
- Hello World!

Public Data Members

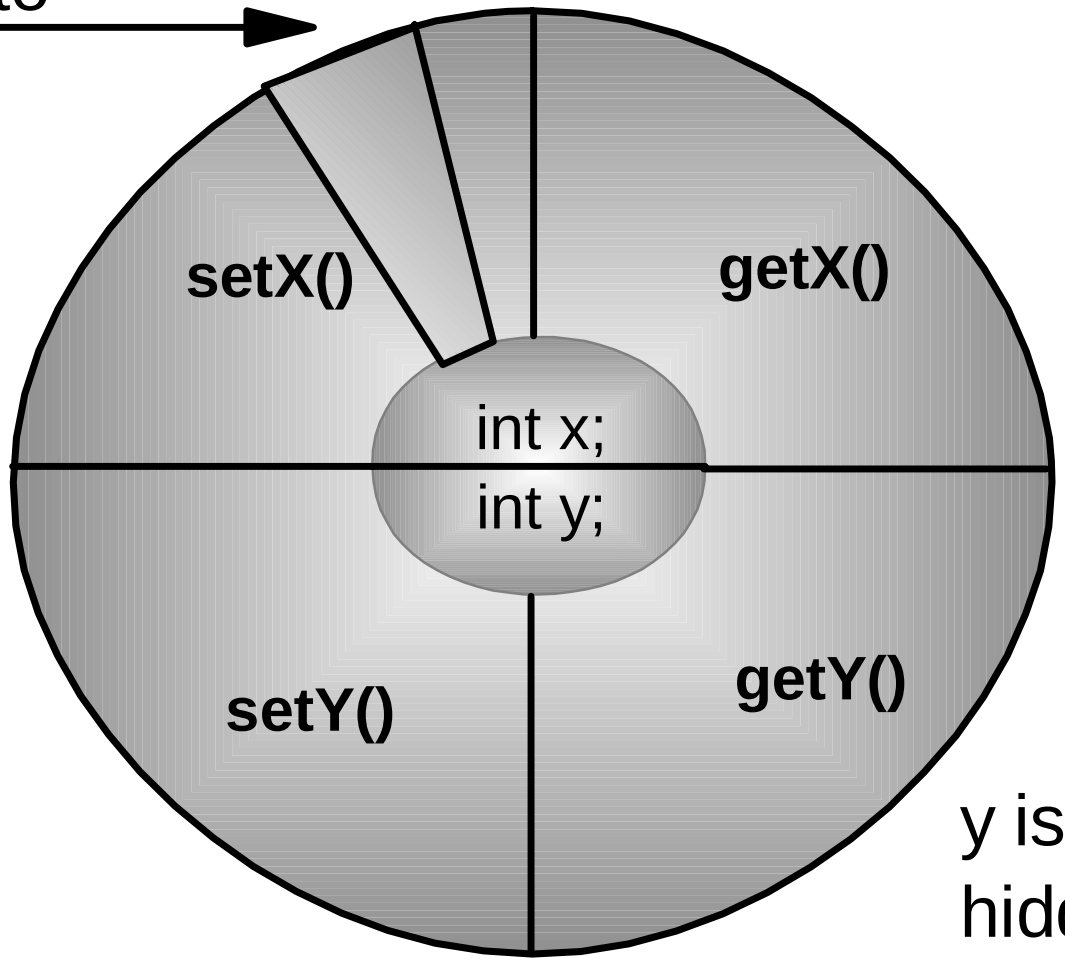
```
class SimplePoint {

  // Data encapsulated by the class
  private int x;
  private int y;

  // Methods that form external interface
  public int getX() { return x; }
  public int getY() { return y; }
  public void setX(int pos) { x = pos; }
  public void setY(int pos) { y = pos; }
}
```

Public Data Members

```
class SimplePoint2 {

  // Data made available by the class
  public int x;

  // Data encapsulated by the class
  private int y;

  // Methods that form external interface
  public int getX() { return x; }
  public int getY() { return y; }
  public void setX(int pos) { x = pos; }
  public void setY(int pos) { y = pos; }
}
```

SimplePoint2

Direct access to x!



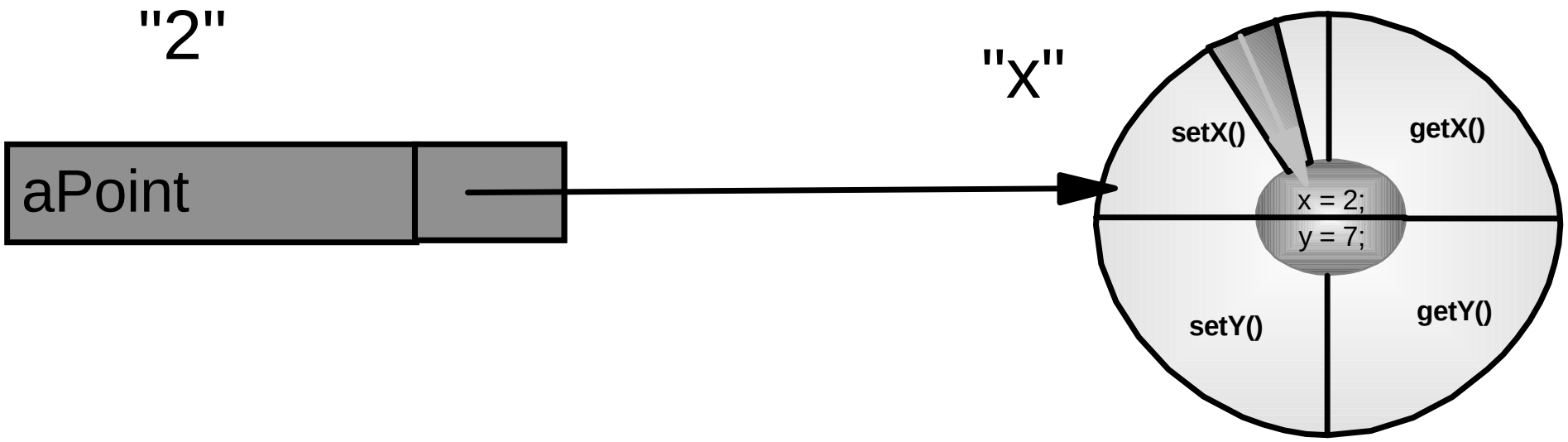
y is private so still hidden

Public Data Members

- Break encapsulation
- Accessed via dot operator
  - aPoint.x is the value of x inside aPoint
  - aPoint.x = 5; sets aPoints x data member to 5
- Use sparingly!

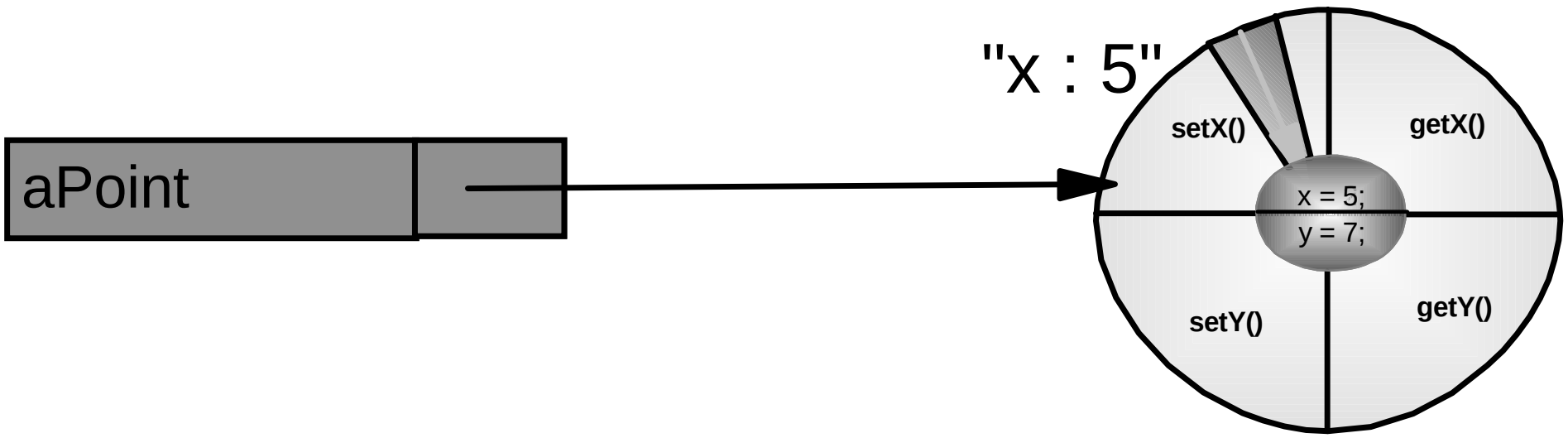
Pictorially...

- if (aPoint.x > 5) { ...



Pictorially...

- aPoint.x = 5;



Agenda

- Introduction to classes and objects
- The lifecycle of an object
- Public data members
- Class data and methods
- Hello World!

Classes revisited

- Each SimplePoint object has its own copy of x and y
- What if we wanted to share some data across all SimplePoint objects?
- We need some *class* data (as opposed to *instance* data)

Classes revisited

- Instance data is dynamic, it is created and destroyed along with the objects that contain it
- Class data is static - there is only one copy associated with the class

Class (static) data

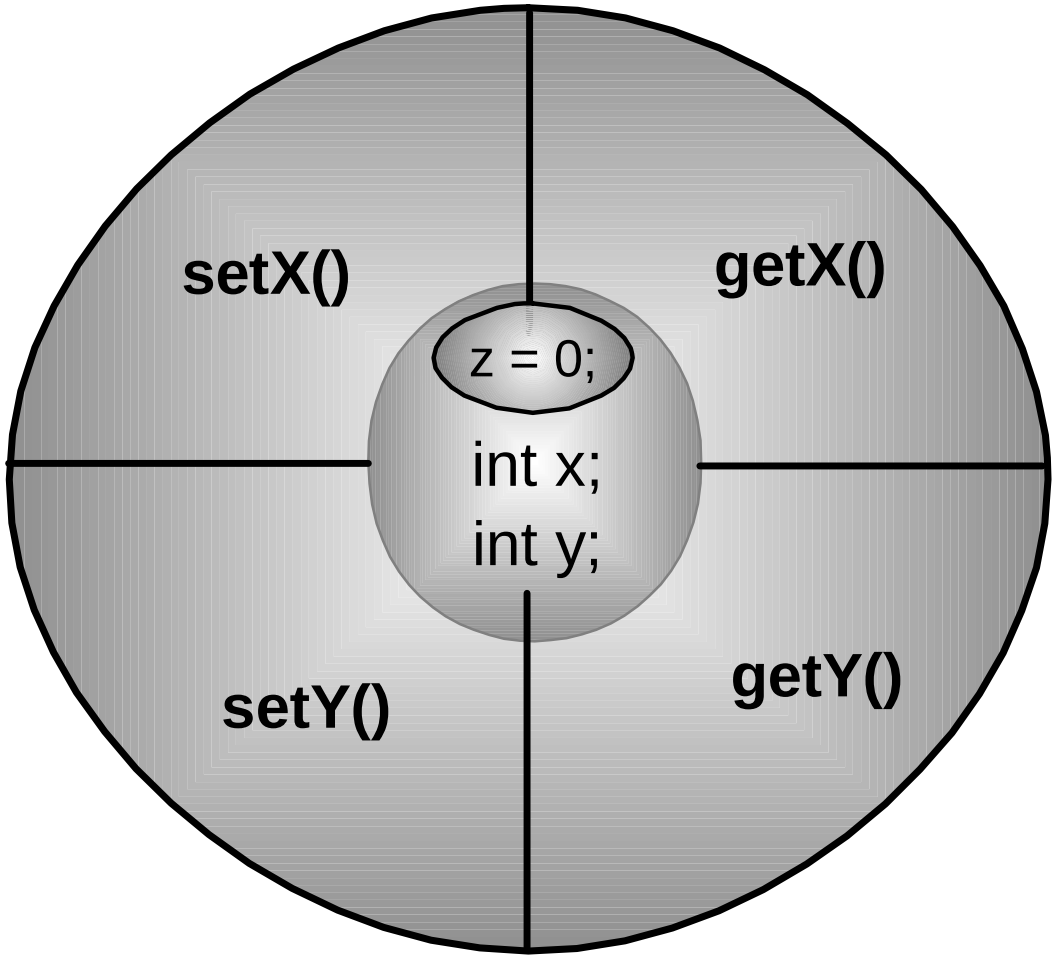
```
class SimplePoint3D {

  // Class data (shared by all SimplePoint3D objs)
  private static int z;
  // Data encapsulated by the class
  private int x;
  private int y;

  // Methods that form external interface
  public int getX() { return x; }
  public int getY() { return y; }
  public void setX(int pos) { x = pos; }
  public void setY(int pos) { y = pos; }
}
```

Class (static) data

- With class data, a class is more than just a type
- Static data is associated with a *class object*
- Java creates the class object the first time the class is used in the program

SimplePoint3D

The single class object keeps the static data.

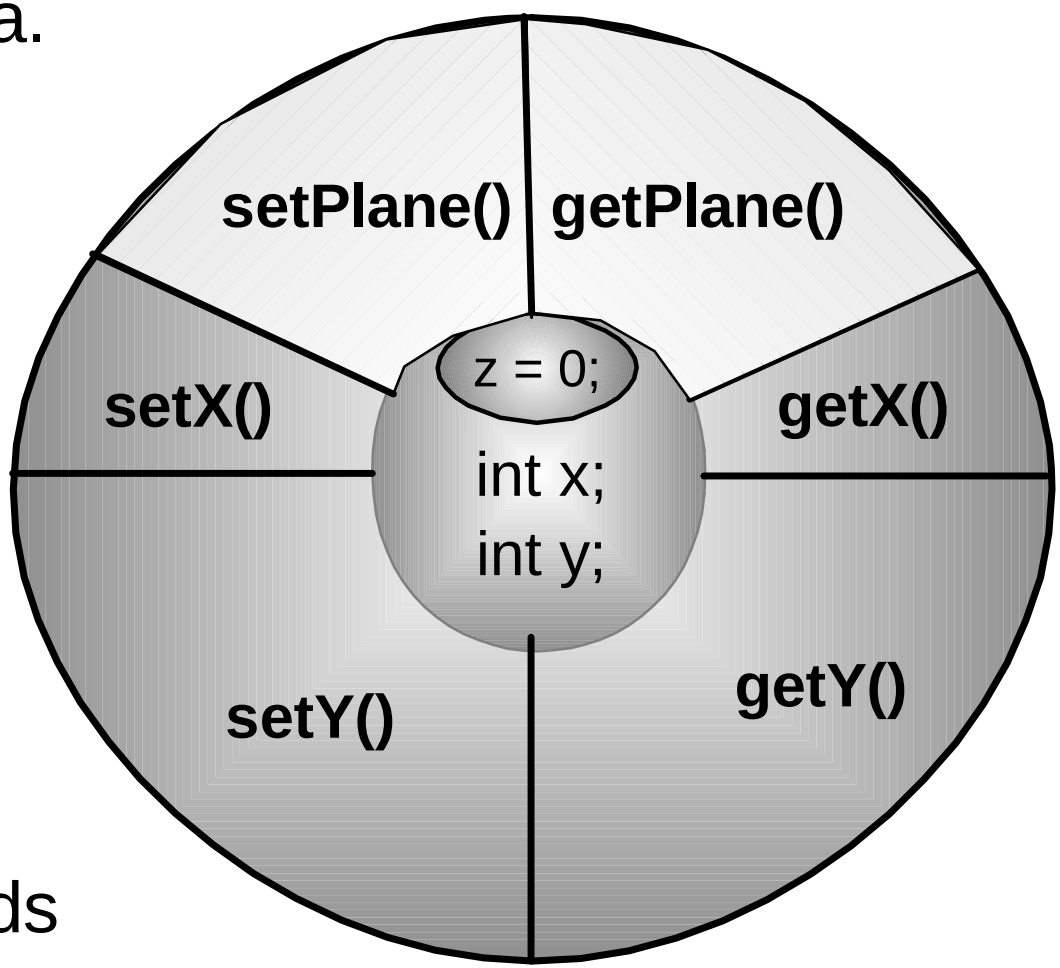


Class methods

- Similarly, static methods may be defined.
  - **public static void** setPlane(**int** pos) { z = pos;}
  - **public static int** getPlane() { **return** z;}
- The class object is referred to simply by the name of the class
  - SimplePoint3D.setPlane(-5);
  - SimplePoint3D.getPlane();

SimplePoint3D

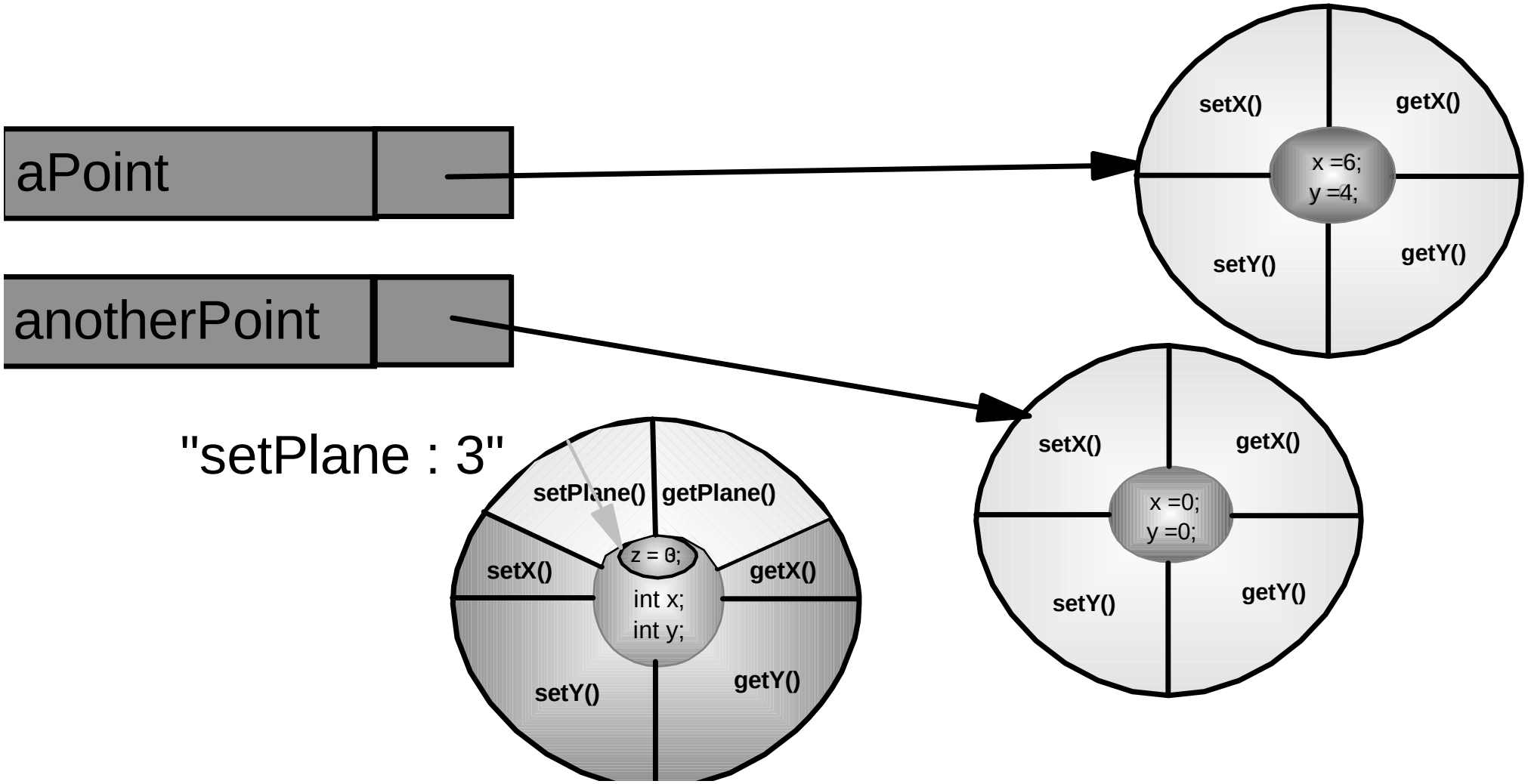
The single class object keeps the static data.



Static methods access the static data.

Class Methods

- SimplePoint3D.setPlane(3);



Class Methods

- To use class methods (and data), you do not need to create any instances of the class, you simply send messages directly to the class object itself.

Agenda

- Introduction to classes and objects
- The lifecycle of an object
- Public data members
- Class data and methods
- Hello World!

Hello World


```
/**
 * The HelloWorldApp class implements an application that
 * simply displays "Hello World!" to the standard output.
 */
class HelloWorldApp {
    public static void main(String[] args) {
        System.out.println("Hello World!"); //Display the string.
    }
}
```

Hello World

class HelloWorldApp {

- *declares a new class called HelloWorldApp*

public static void main(String[] args) {

- *class method called main (invoked by interpreter as HelloWorldApp.main, no need to create an instance of the class)*

System.out.println("Hello World!"); //Display the string.

- *System is a class defined in the Java API. Since we have not created any instances of the System class, we can deduce that out must be a public data member*
- *The "out" data member is of type (class) PrintWriter, so the expression System.out evaluates to an instance of the PrintWriter class*
- *PrintWriter has a method called println, which prints a line of text!*

}

}

Summary

- OO is an evolution, not a revolution
- Classes are like user defined types
- Objects are like variables of those types
- We send messages which invoke methods
- Classes have a class object

Class Design Hints

1. Always keep data private
2. Always initialize data
3. Don't use too many basic types in a class
4. Not all fields need individual accessor and mutator (get and set) methods
5. Break up classes with too many responsibilities
6. Make the names of classes and methods reflect their responsibilities

What you don't know!

- Access modifiers (public, private, protected, none)
- Finalizers
- Method overriding
- Data hiding
- Constructor chaining
- Equality testing
- Abstract classes and methods
- Interfaces

What you do know

- Is enough to give you a solid base from which to explore these ideas...