Java Refresher

Alexandre Bergel
http://bergel.eu
14/03/2018

Goal of this lecture

This lecture will essentially be a *Java refresher*

Understand *what an object is*

Understanding *some important* particularities of Java

Outline

1. Java refresher

   1. Small illustrative scenario with the class Point

   2. Extending Point into PositivePoint

2. Class inheritance

3. Terminology

Outline

1.Java refresher

    1.Small illustrative scenario with the class Point

    2.Extending Point into PositivePoint

2.Class inheritance

3.Terminology

Defining point as a first example...

We will model points as a first example

A point, is a particular entity that represents a spacial location

It contains two values, x and y

Defining point as a first example...

It answers to four messages

    moveTo(newX, newY) - move the point to a new location

    moveBy(deltaX, deltaY) - incrementally move the point

    getX() - return the X value

    getY() - return the Y value

We will now play with one point

What do we need next?

We know the *principal character* (the point), but this is *not enough to make a movie*! We still need:

To determine where the point comes from

Point will be the name of the factory

An execution scenario

PointExample will be the name of the scenario

A simple execution scenario

It may not be as good a Hollywood movie scenario, but it will be shorter:

1 - PointExample creates the hero of our story, a point called myLittlePoint.

2 - PointExample tells to the World where myLittlePoint is

3 - myLittlePoint, who wants to discover the World, jumps to a position

4 - He/She/It then does a little step

5 - End of the movie!

Screen-cast

Point

The factory

PointExample

The scenario

...

The hero

...

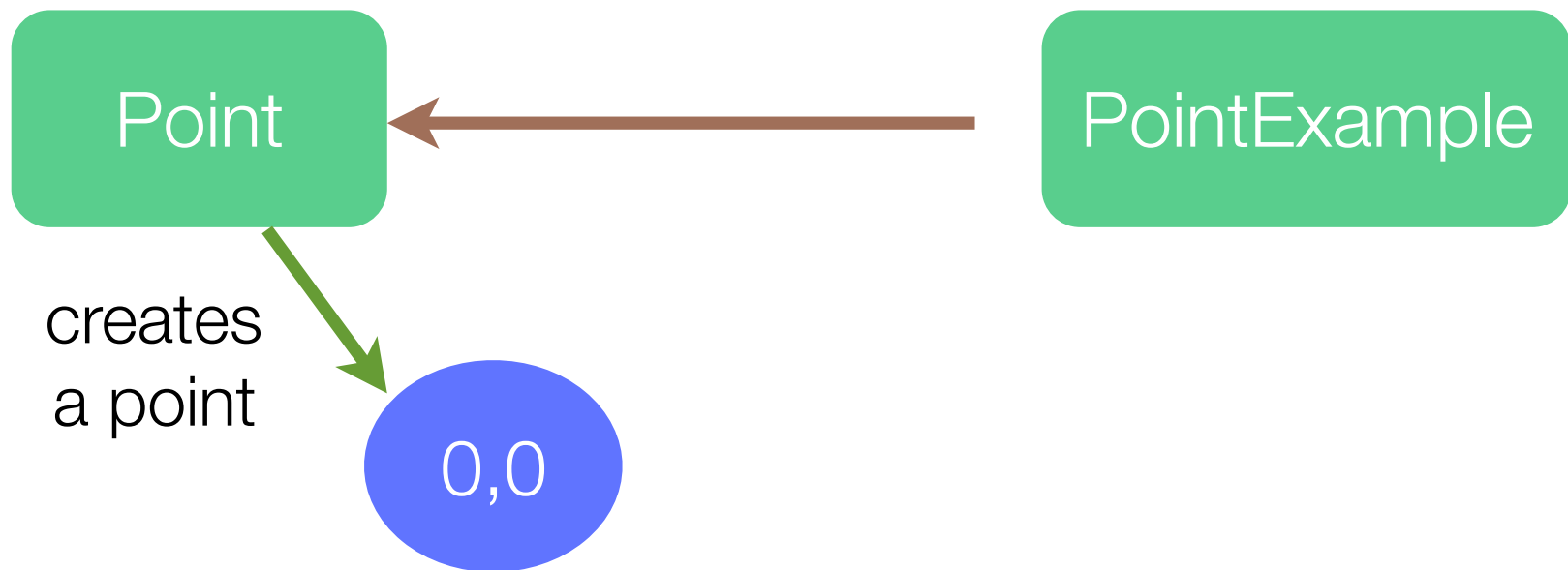System.out

A simple execution scenario

Point

PointExample

PointExample asks the factory to create our hero

Point ← create a point! — PointExample

PointExample asks the factory to create our hero

Point ← PointExample

creates
a point

0,0

PointExample gives a name to our hero

PointExample

myLittlePoint

0,0

PointExample gives a name to our hero

PointExample

myLittlePoint

0,0

The name "myLittlePoint" is known only by PointExample
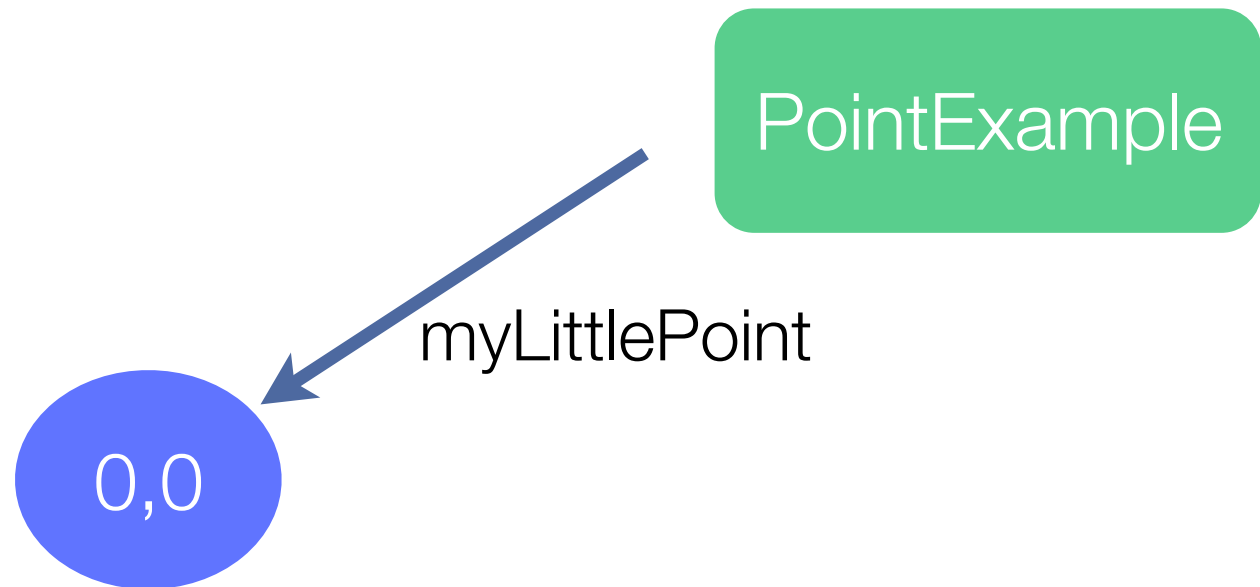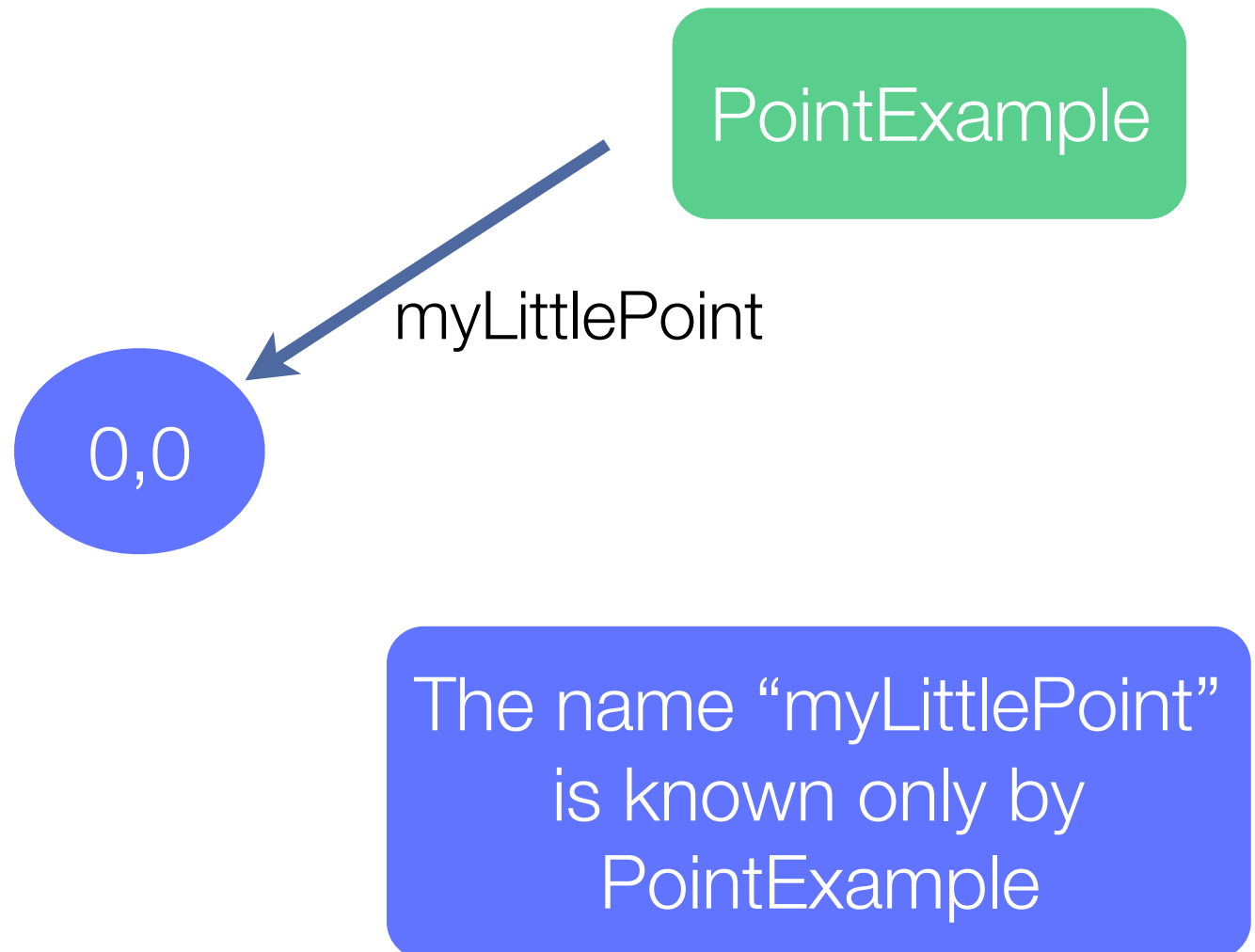
PointExample says to the World where our hero is

PointExample

myLittlePoint

0,0

print(...)!

System.out   ...

`Initial position is 0,0`
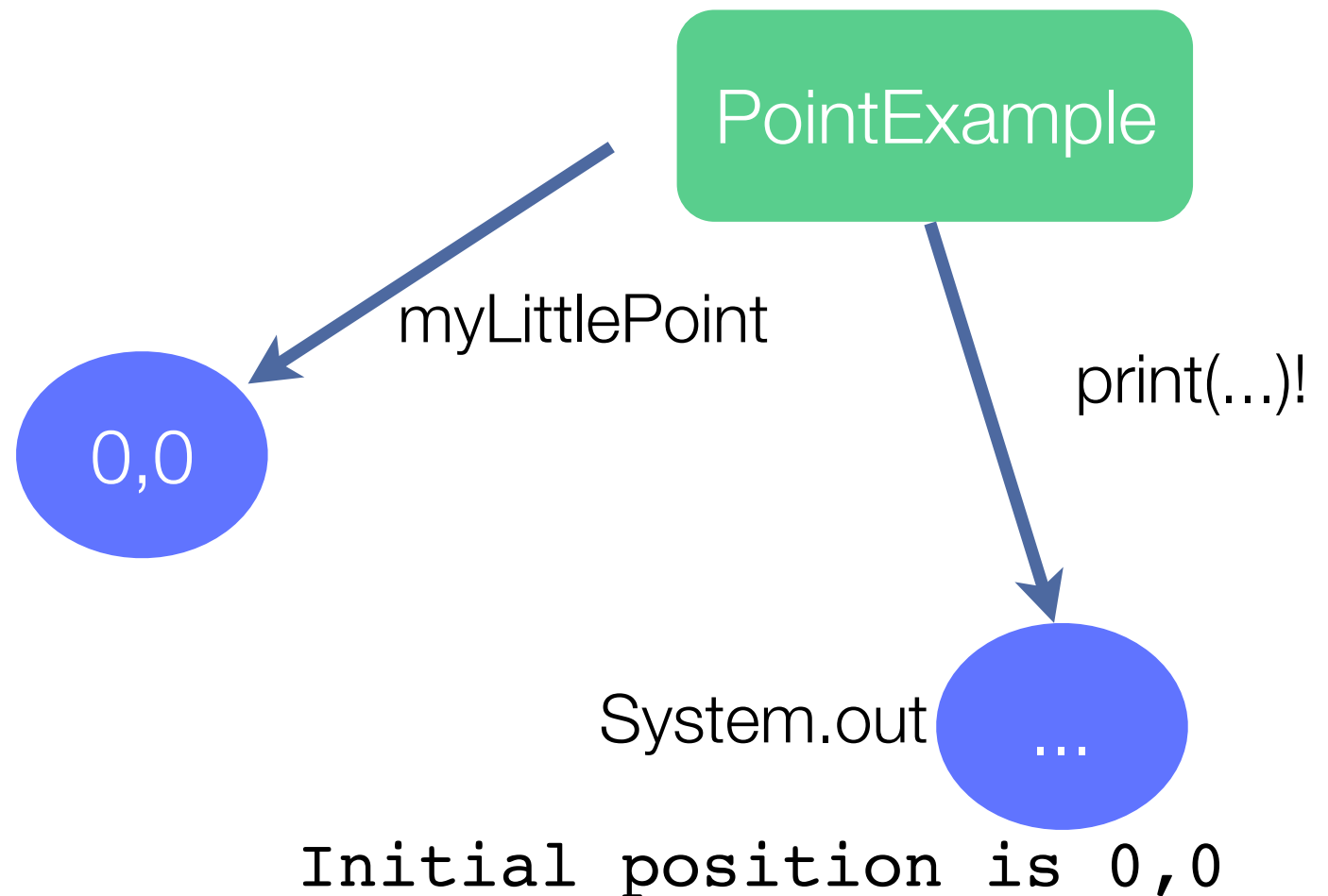
PointExample asks our hero to jump to a new location

PointExample

myLittlePoint,
move to 4,2!

4,2
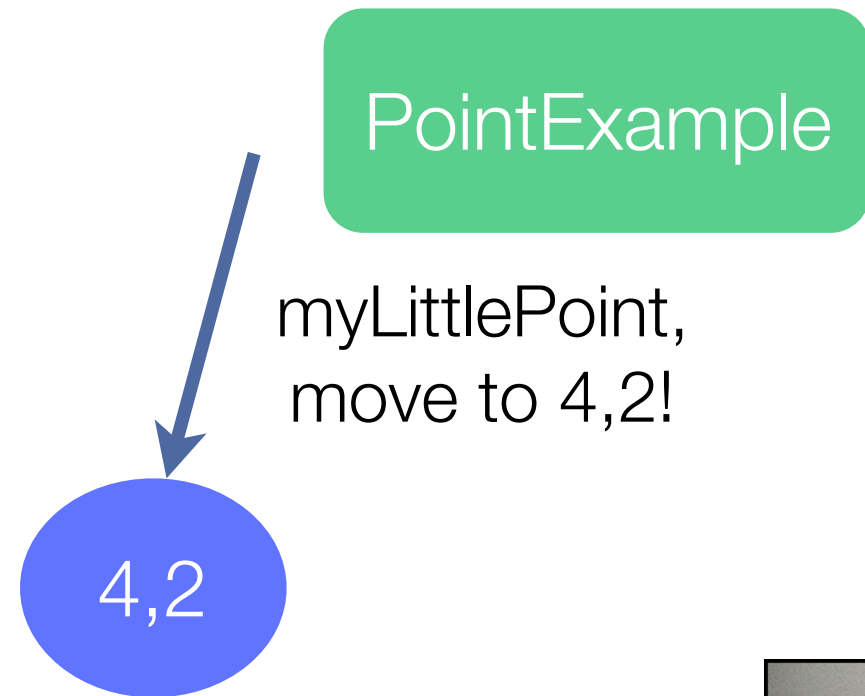
... say to the World where the hero is...

PointExample

myLittlePoint, move to 4,2!

print(...)!

4,2

System.out

...
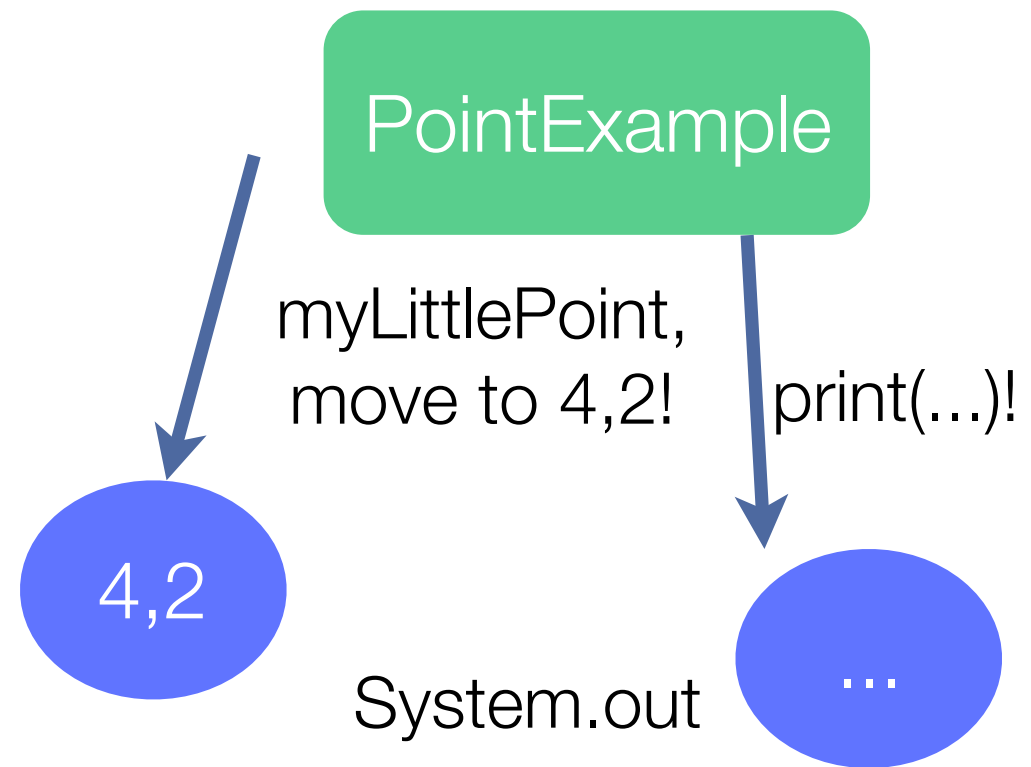
```
Initial position is 0,0
Position after a moveTo is 4,2
```

... now a little step...

PointExample

myLittlePoint,
move by 1,3!

5,5
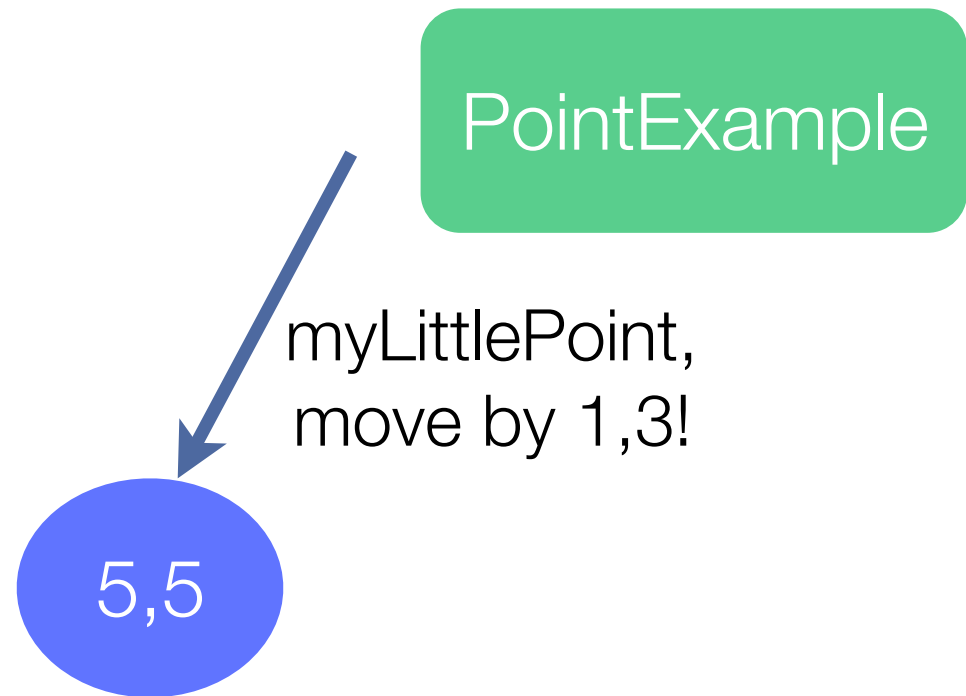
... and finally, tell to the World where our hero is!

PointExample

myLittlePoint, move by 1,3!

print(...)!

5,5

...

System.out
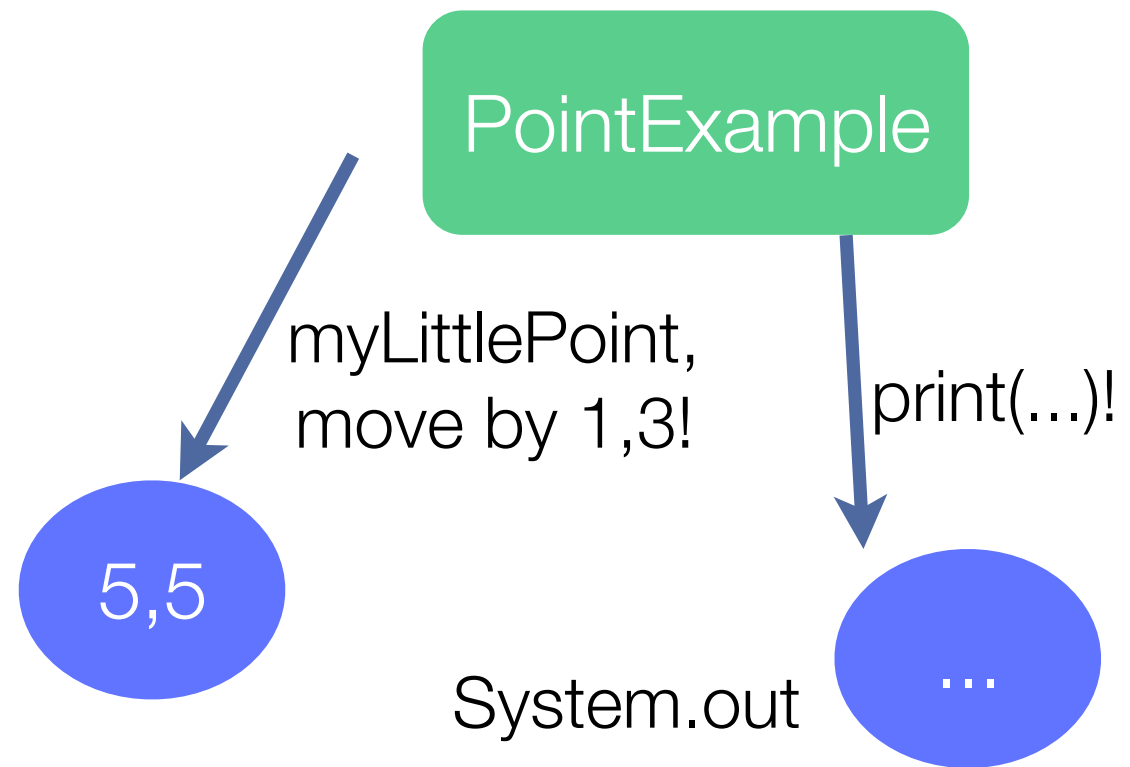
```
Initial position is 0,0
Position after a moveTo is 4,2
Position after a moveBy 5,5
```

Some of the important parts that you should not have missed! ...

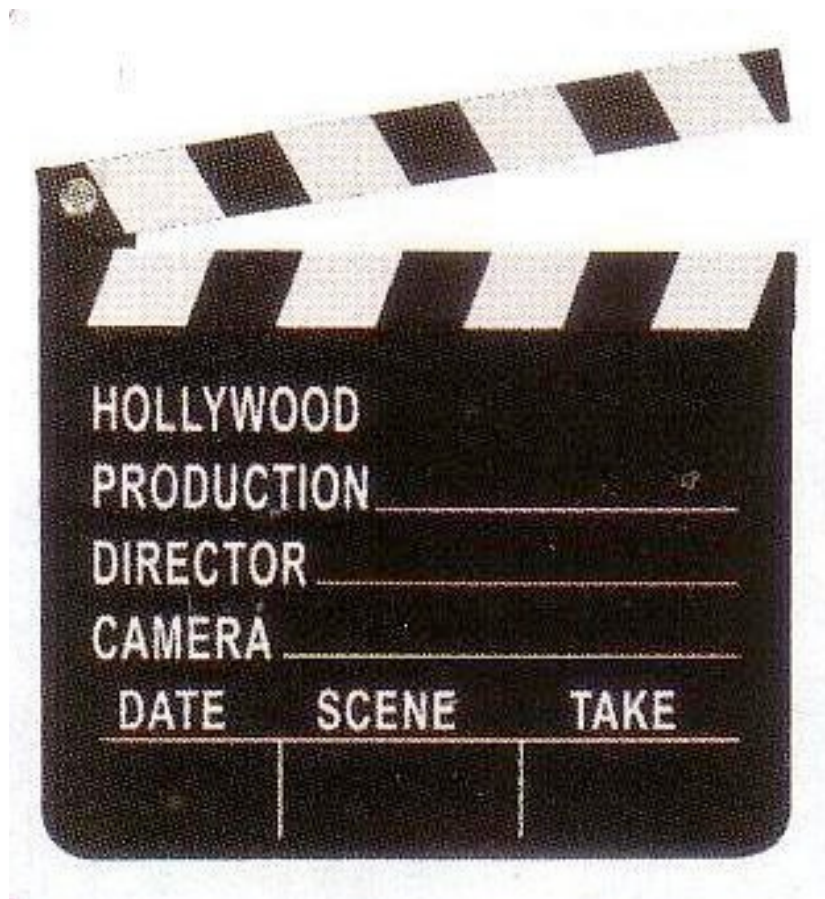Point is the only character who knows what myLittlePoint looks like and how it behaves

Point knows how to interpret the orders given to myLittlePoint

myLittlePoint only knows

the value of X and the value of Y

who created it

Only PointExample knows that the hero is named myLittlePoint

Other character, if any, can call the hero as they wished

Some of the important parts that you should not have missed!

PointExample *sends* to myLittlePoint some orders, defined in term of *messages*

PointExample cannot move myLittlePoint directly (by changing the value of x and y), *it has to ask myLittlePoint* to do it

Only myLittlePoint can do the job to move to a new location

PointExample *cannot send a message* to myLittlePoint that is not understood

A first example, the code
Definition of Point

```
package cc3002;

public class Point {

    protected int x;

    protected int y;

    ...
```

Initialize point creation

```
...

public Point() {

    x = 0;

    y = 0;

}

...
```

moveTo and moveBy

```
...

public void moveTo(int newX, int newY) {

    x = newX;

    y = newY;

}

public void moveBy(int deltaX, int deltaY) {

   x = x + deltaX;

   y = y + deltaY;

}

...
```

getting the location

```
   ...
  public int getX() {

     return x;

  }

  public int getY() {

     return y;

  }

}
```

Definition of PointExample

```
package cc3002;

public class PointExample {

    public static void main(String[] argv) {

      Point myLittlePoint = new Point();

      System.out.println("Initial position " +

            myLittlePoint.getX()+ ","+ myLittlePoint.getY());

      myLittlePoint.moveTo(4, 2);

      System.out.println("Position after a moveTo " +

            myLittlePoint.getX()+ ","+ myLittlePoint.getY());

       ...
```

Magic invocation, we will see that later

```java
package cc3002;

public class PointExample {
    public static void main(String[] argv) {

    Point myLittlePoint = new Point();

    System.out.println("Initial position " +
         myLittlePoint.getX()+ ","+ myLittlePoint.getY());

    myLittlePoint.moveTo(4, 2);

    System.out.println("Position after a moveTo " +
         myLittlePoint.getX()+ ","+ myLittlePoint.getY());

     ...
```

Entry point of the program

Creation of an unnamed point

```
package cc3002;

public class PointExample {

    public static void main(String[] argv) {

        Point myLittlePoint = new Point();

        System.out.println("Initial position " +

            myLittlePoint.getX()+ ","+ myLittlePoint.getY());

        myLittlePoint.moveTo(4, 2);

        System.out.println("Position after a moveTo " +

            myLittlePoint.getX()+ ","+ myLittlePoint.getY());

         ...
```

Give the name of the new point

```java
package cc3002;

public class PointExample {

    public static void main(String[] argv) {

        Point myLittlePoint = new Point();

        System.out.println("Initial position " +
                myLittlePoint.getX()+ ","+ myLittlePoint.getY());

        myLittlePoint.moveTo(4, 2);

        System.out.println("Position after a moveTo " +
                myLittlePoint.getX()+ ","+ myLittlePoint.getY());

        ...
```

Give the name of the new point

```java
package cc3002;

public class PointExample {

    public static void main(String[] argv) {

        Point myLittlePoint = new Point();

        System.out.println("Initial position " +

            myLittlePoint.getX()+ ","+ myLittlePoint.getY());

        myLittlePoint.moveTo(4, 2);

        System.out.println("Position after a moveTo " +

            myLittlePoint.getX()+ ","+ myLittlePoint.getY());

        …
```

Again, the object may have a different name in a different context

Ask to myLittlePoint for its position

```
package cc3002;

public class PointExample {

    public static void main(String[] argv) {

        Point myLittlePoint = new Point();

        System.out.println("Initial position " +
            myLittlePoint.getX()+ ","+ myLittlePoint.getY());

        myLittlePoint.moveTo(4, 2);

        System.out.println("Position after a moveTo " +
            myLittlePoint.getX()+ ","+ myLittlePoint.getY());

        ...
```

Show it in a standard output stream

```
package cc3002;

public class PointExample {

    public static void main(String[] argv) {

      Point myLittlePoint = new Point();

      System.out.println("Initial position " +

          myLittlePoint.getX()+ ","+ myLittlePoint.getY());

      myLittlePoint.moveTo(4, 2);

      System.out.println("Position after a moveTo " +

          myLittlePoint.getX()+ ","+ myLittlePoint.getY());

       ...
```

Ask the hero to move to a new location

```java
package cc3002;

public class PointExample {

    public static void main(String[] argv) {

        Point myLittlePoint = new Point();

        System.out.println("Initial position " +

            myLittlePoint.getX()+ ","+ myLittlePoint.getY());

        myLittlePoint.moveTo(4, 2);

        System.out.println("Position after a moveTo " +

            myLittlePoint.getX()+ ","+ myLittlePoint.getY());

        ...
```

Ask the location once more

```
package cc3002;

public class PointExample {

    public static void main(String[] argv) {

        Point myLittlePoint = new Point();

        System.out.println("Initial position " +
                myLittlePoint.getX()+ ","+ myLittlePoint.getY());

        myLittlePoint.moveTo(4, 2);

        System.out.println("Position after a moveTo " +
                myLittlePoint.getX()+ ","+ myLittlePoint.getY());

        ...
```

Print the result in the output standard stream

```java
package cc3002;

public class PointExample {

    public static void main(String[] argv) {

        Point myLittlePoint = new Point();

        System.out.println("Initial position " +

                myLittlePoint.getX()+ ","+ myLittlePoint.getY());

        myLittlePoint.moveTo(4, 2);

        System.out.println("Position after a moveTo " +

                myLittlePoint.getX()+ ","+ myLittlePoint.getY());

        ...
```

myLittlePoint is asked to move by 1,3

```
myLittlePoint.moveBy(1, 3);

System.out.println("Position after a moveBy " +

        myLittlePoint.getX()+ ","+ myLittlePoint.getY());

    }

}
```

The location is asked once more

```
myLittlePoint.moveBy(1, 3);

System.out.println("Position after a moveBy " +
    myLittlePoint.getX()+ ","+ myLittlePoint.getY());

}

}
```

The position is displayed

```
myLittlePoint.moveBy(1, 3);

System.out.println("Position after a moveBy " +

    myLittlePoint.getX()+ ","+ myLittlePoint.getY());

}

}
```

Running the example

```
> java -cp bin cc3002.PointExample

My initial position 0,0

My position after a moveTo 4,2

My position after a moveBy 5,5
```

Java particularities

Java is a *class-based object-oriented language*

... but not completely

a *class instantiation is not done through message sending*, but with an *operator*

Java contains primitive types, which are not objects

Static methods are not looked up

only methods (or also called instance methods) that are not private are looked up

we will come back on that point in the future

E.g., the main() method is called directly by the VM, without instantiating PointExample

Defining a positive point...

```
package cc3002;


public class PositivePoint extends Point {

  public void moveBy(int deltaX, int deltaY) {

     super.moveBy(deltaX, deltaY);

     x = (x < 0) ? 0 : x;

     y = (y < 0) ? 0 : y;

  }

     ...
```

Defining a positive point

```
    ...

  public void moveTo(int newX, int newY) {

      if (newX < 0) { newX = 0; }

      if (newY < 0) { newY = 0; }

      super.moveTo(newX, newY);

  }

}
```

Outline

1.Java refresher
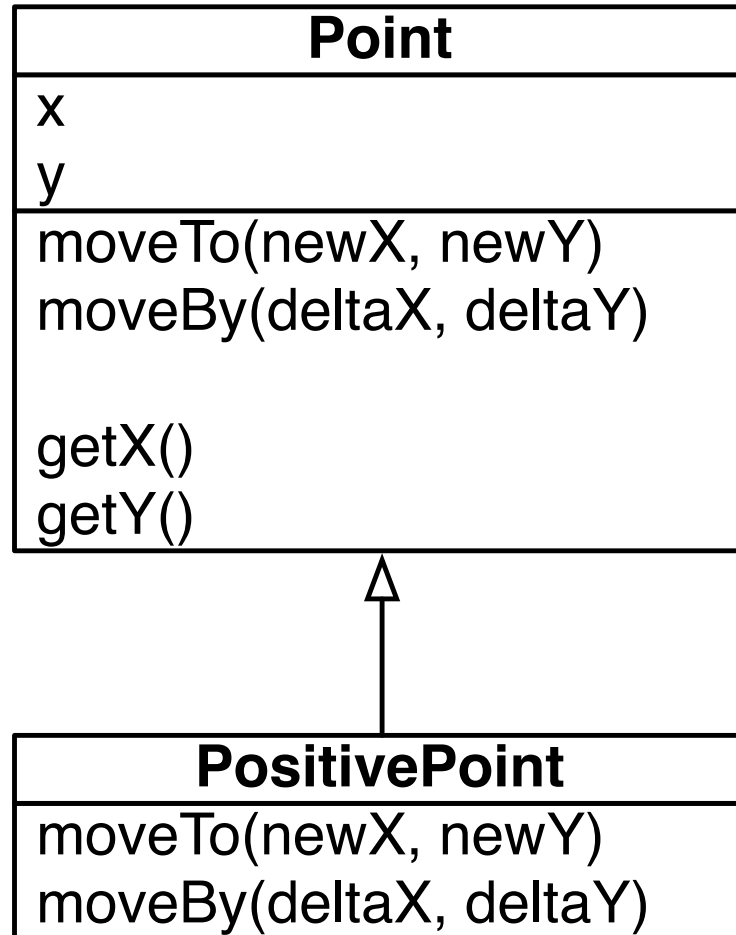
   1.Small illustrative scenario with the class Point

   2.Extending Point into PositivePoint

2.Class inheritance

3.Terminology
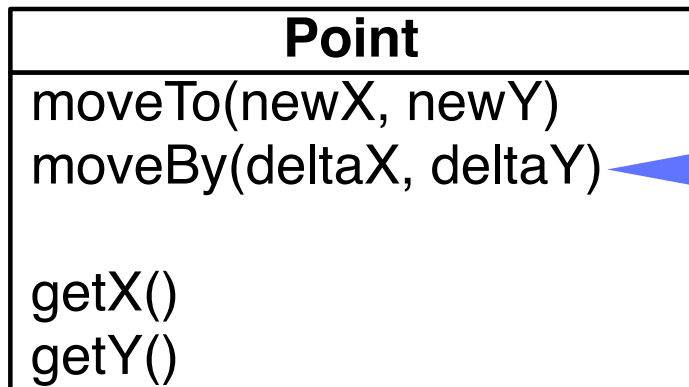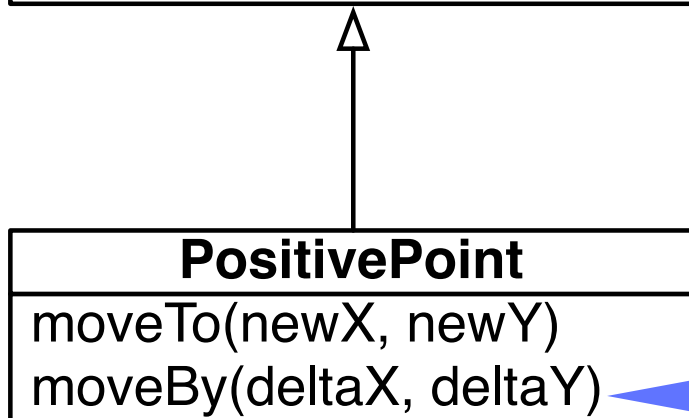
Class inheritance

Point
x
y
moveTo(newX, newY)   moveBy(deltaX, deltaY)   getX()   getY()

PositivePoint
moveTo(newX, newY)   moveBy(deltaX, deltaY)

Defining a positive point

Point
moveTo(newX, newY)
moveBy(deltaX, deltaY)
getX()
getY()

x = x + deltaX;
y = y + deltaY;

PositivePoint
moveTo(newX, newY)
moveBy(deltaX, deltaY)

super.moveBy(dX, dY);
x = (x < 0) ? 0 : x;
y = (y < 0) ? 0 : y;

Class inheritance

Point
moveTo(newX, newY) moveBy(deltaX, deltaY) getX() getY()
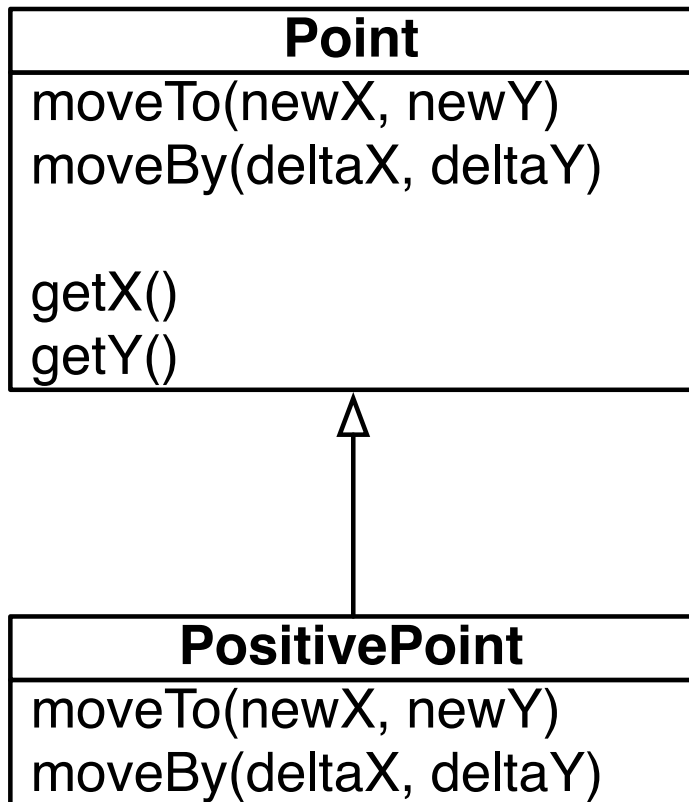
```
Point aPoint = new Point();
aPoint.moveTo(4, 2);

=> execute Point.moveTo(...)
```

PositivePoint
moveTo(newX, newY) moveBy(deltaX, deltaY)

Class inheritance

Point
moveTo(newX, newY) moveBy(deltaX, deltaY) getX() getY()

PositivePoint
moveTo(newX, newY) moveBy(deltaX, deltaY)

`aPoint = new PositivePoint();`

aPoint

instantiation

Class inheritance

Point
moveTo(newX, newY) moveBy(deltaX, deltaY) getX() getY()

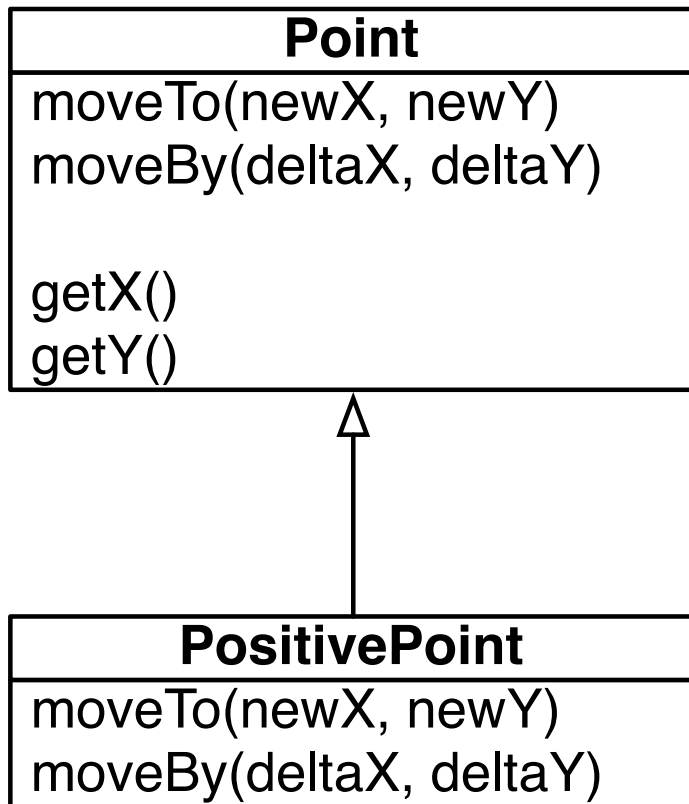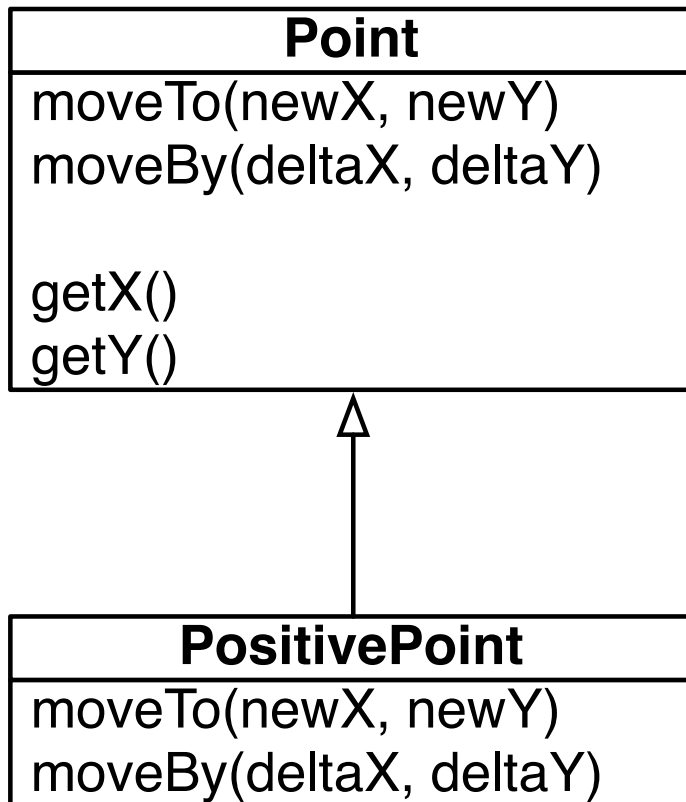| **PositivePoint** |
| --- |
| moveTo(newX, newY)
moveBy(deltaX, deltaY) |
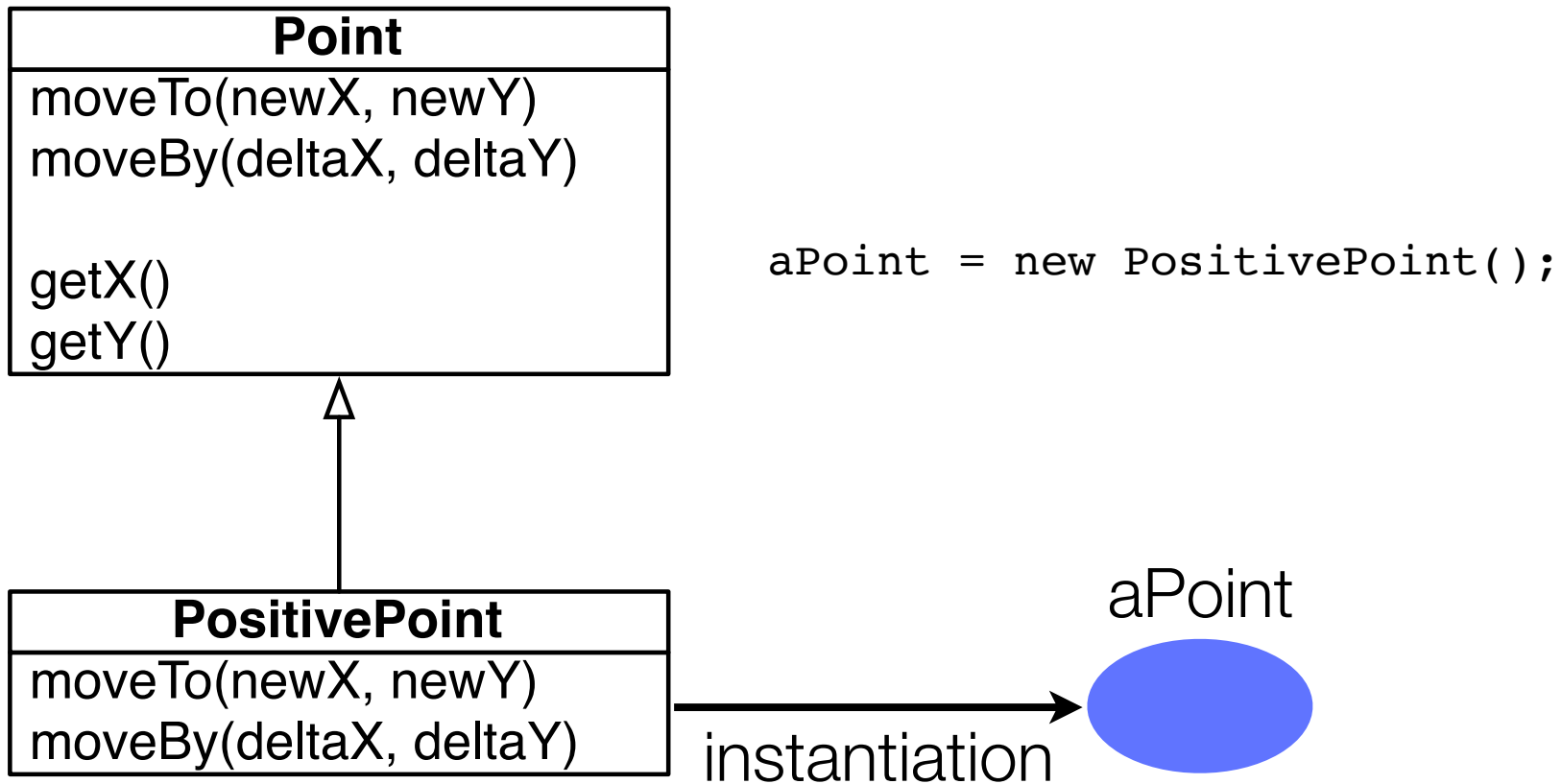
```
Point aPoint = new PositivePoint();
aPoint.moveTo(4, 2);

=> execute PositivePoint.moveTo(...)
```

49

Class inheritance

Point
moveTo(newX, newY)
moveBy(deltaX, deltaY)
getX()
getY()

```
Point aPoint = new PositivePoint();
aPoint.getX();

=> execute Point.getX(...)
```

PositivePoint
moveTo(newX, newY)
moveBy(deltaX, deltaY)

50

Class inheritance

During the first weeks of the semester we will explain *how* inheritance works

However, understand *when* to use inheritance is the topic of the whole semester

Class inheritance is highly powerful:

It may bring fantastic property regarding extensibility in a software system

But it may be devastating if not properly used

Outline

1. Java refresher

    1. Small illustrative scenario with the class Point

    2. Extending Point into PositivePoint

2. Class inheritance

3. **Terminology**

Terminology

Object

"An object is a software machine allowing programs to access and modify a collection of data" -- *Class of Touch, Bertrand Meyer*

"Objects are not just simple bundles of logic and data. They are responsible members of an object community" -- *Object Design, Rebecca Wirfs-Brock and Alan McKean*

An object has a *unique position in memory*, often assimilated as its identify

An object knows from which class it has been created from (for class-based object-oriented programming languages like Java, C#, Smalltalk)

An object *understands* the *messages* for the methods inherited and defined in its class

Terminology

Class

A *class is primarily an object factory*

It is defined as a set of variable declarations and method definitions

Conceptually: *class = name + variables + methods + superclass*

In Java: *class = name + variables + methods + superclass + interfaces + static methods + ...*

Terminology

Method

Executable piece of code

A method execution ends (i) when no more instruction have to be executed; (ii) when a return statement is reached; (ii) when an exception is raised

The control flow is returned to its caller method when the method return

Can access to the *this* and *super* pseudo-variables (only in instance method; cannot be used in a static method in Java)

Terminology

Inheritance

relation of *specialization* between classes

a subclass *inherits attributes* and *behavior* from its superclass

it is considered *bad programming style* to use inheritance for code *reuse only*

Terminology

Polymorphism

is the ability of one type A to appear as and be used like another type B

polymorphism plays a key difference between message sending and function invocation

```
Point aPoint = new PositivePoint();
```

What you should know!

What is the difference between an *object* and *class*?

What is the difference between *class reuse* and *class specialization*?

The difference between *function invocation* and *sending messages*

The difference between a *pointer* (à la C/C++) and a *reference*

Can you answer these questions?

Why do objects *"send messages"* instead of *"executing methods"*?

Can you imagine an object model in which a *class is also an object*?

Why *polymorphism* and *class inheritance* are so tightly related in Java?

License

http://creativecommons.org/licenses/by-sa/2.5