Implicit Parameter

Instance methods and Class methods

Example: translate method of Rectangle

- **rect1.translate(100, 80);**
  moves the rectangle 100 units (pixels) to the right and 80 units (pixels) downward
- **Rectangle rect1 = new Rectangle(70,90,100,150);**
- **rect1.translate(100, 80);**
- **moves the rectangle to be a rectangle with the data (170, 170, 100, 150)**
- ***Instance methods*** **are those that have to be called on an object (the other methods are *static* methods, also called *class* methods, e.g Math.max)**

Getting to the issue of implicit parameter

```
public static void main(String[ ] args) {
    Rectangle rect1 = new Rectangle(70,90,100,150);
    Rectangle rect2 = new Rectangle(70,180,200,50);
    rect1.translate(100, 80);
}
```

- There is only one copy of the code for the method *translate*, not one copy for each object
- How is that *translate* moves rect1 and not rect2 ?
- The code of *translate* has to receive a reference to the object it will read information from or modify
- Compile and run the code of FirstGUI.java

The reference is passed as if it were a parameter

- The activation record of a method contains various parts starting with the space designated for the parameters, where the arguments are stored
- For instance methods, the reference to the object is also stored in the activation record as if it were a parameter.
- It is called the *implicit parameter* (Section 3.7)

Putting the implicit parameter into the stack

- The rect1.translate(dx, dy) call causes the values of
  rect1
  dx
  dy

  to be put into the activation record in the call stack.
- The implicit parameter carries the name “this”
- The code executed is then

```
rect1.x = rect1.x + dx;
rect1.y = rect1.y + dy;
```

The presence of this is understood

- The code of the translate method can be written as

```
x = x + dx;
y = y + dy;
```

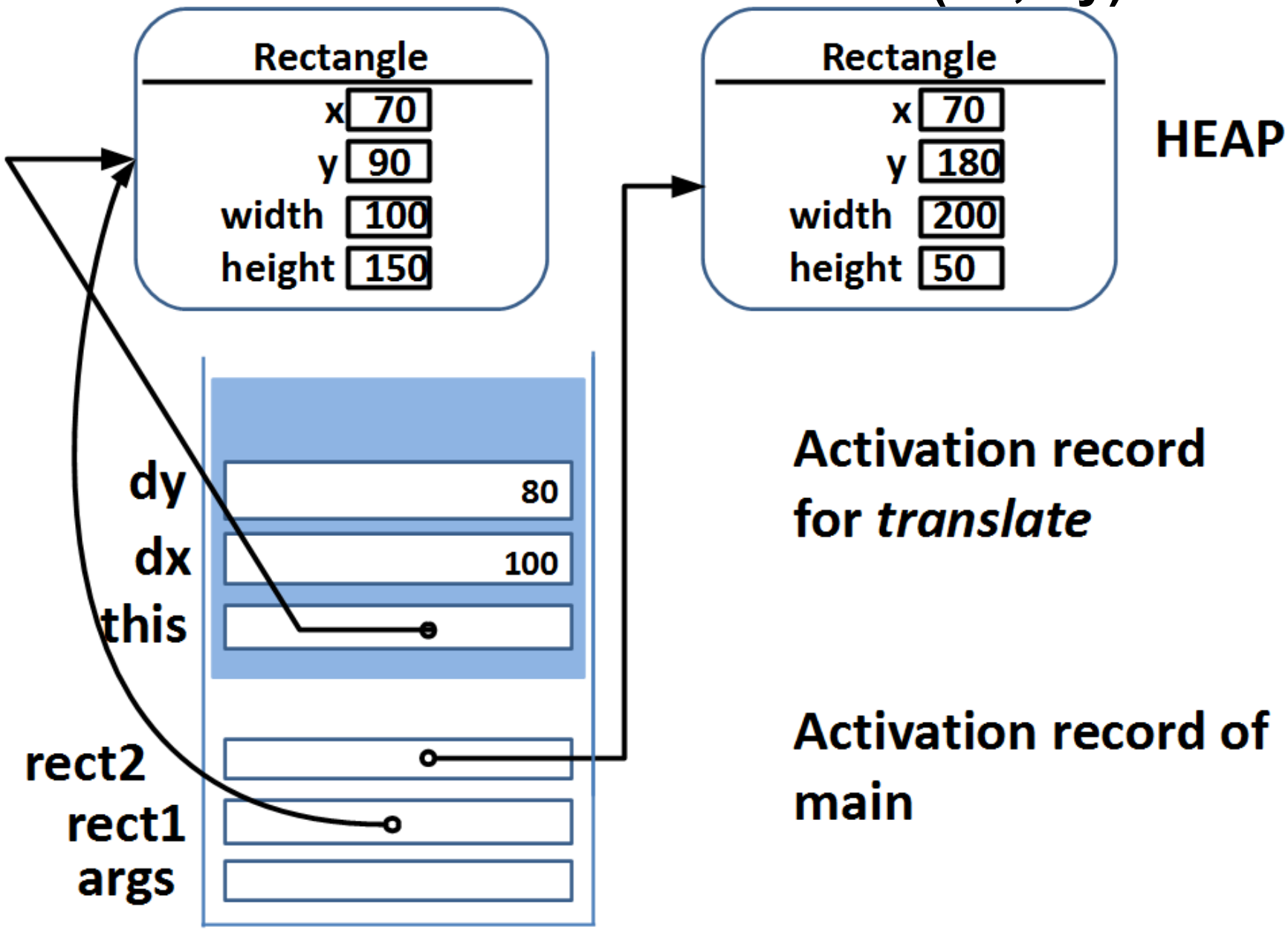
or

```
this.x = this.x + dx;
this.y = this.y + dy;
```

- They are synonymous for the compiler. Hence if the current value of “this” is the reference “rect1” then the code executed is indeed

```
rect1.x = rect1.x + dx;
rect1.y = rect1.y + dy;
```

Activation record for translate(dx, dy)



Instance vs class methods

- If the method needs to read or modify the instance fields (i.e. fields that are not declared as static), then the code needs access to the reference “this” which we call the implicit parameter
- If the method is more stand-alone as was the case with the methods we saw for arrays, then they can be declared as *static*. These are also called *class* methods and they do not have the implicit parameter in their activation records