- Implicit parameter
- In Java the object associated with the instantiation of a class is an implicit argument or parameter to the classes' instance methods
  - That is, the instance methods need to know what object they are being called on
  - The static methods of a class have the object explicitly stated as part of the method invocation
- For example, the String class has a length() method
- If we instantiate a String with “String s = new String(“abcdef”);” then execute “s.length();”, the object s of type String is an implicit argument or parameter to the method string()
- If we have a “public static int length(String s)” method, then “s” is an explicit argument or parameter

- Example
  - The Java Rectangle class has instance fields
    - x, y, width, height
  - It also has an instance method translate(int dx, int dy) which translates the values of x & y by dx & dy
  - If we create two Rectangles r1 & r2
    - r1 = (0, 0, 10, 10) and r2 = (0, 0, 20, 20)
  - When we perform r1.translate(10, 10) there is an implicit parameter, with the name “this”, which is a reference to the object r1, that is provided to the translate method (along with the two parameters dx & dy)

- Example (cont)
  - In the r1.translate(int dx, int dy) method call the implicit argument allows the translate method to modify the correct Rectangle object
    - The code for translate can be written as
      - x = x+dy;
      - y = y+dy;

      or
      - this.x = this.x+dy;
      - this.y = this.y+dy;
    - They are synonymous to the compiler
      - And if the current value of “this” is a reference to “r1”, then the code executed is
        - r1.x = r1.x+dy;
        - r1.y = r1.y+dy;

- If a method needs access to read or modify instance fields, then the method needs access to the reference “this”, which is called the implicit parameter
- Class (static) methods do not need the implicit parameter, since they do not reference the instance fields