This Keyword

In Java, there is an important keyword: **this**. It can be used inside methods to refer to the current object. Using **this**, you can access the instance variables without any confusion with parameter names. Let’s see what this means over an example. Given the following class definition:

```java
public class Book {
    private String title;
    private int numPages;

    public Book(String t, int n) {
        title = t;
        numPages = n;
    }

    public void setTitle(String t) {
        title = t;
    }

    public void setNumPages(int n) {
        numPages = n;
    }

    public String getTitle() {
        return title;
    }

    public int getNumPages() {
        return numPages;
    }
}
```

One-letter parameter names are not descriptive. It is usually the convention in Java to use the instance variable name in the corresponding set methods. So, we want to do something like

```java
public void setTitle(String title) {
    title = title;
}

public void setNumPages(int numPages) {
    numPages = numPages;
}
```

There is no compile error in this code. The problem is that, the parameters `title` and `numPages"shadow"` the fields `title` and `numPages`. That is, the statement

```java
    title = title;
```

assigns the parameter `title` to itself, causing no change what so ever. In such cases, you may use the **this** keyword to access the fields, as follows:

```java
public void setTitle(String title) {
    this.title = title;
}

public void setNumPages(int numPages) {
    this.numPages = numPages;
}
```
Now the code works properly. You can do the same thing in the constructor, and to be consistent, in the get methods, as well:

```java
public class Book {
    private String title;
    private int numPages;

    public Book(String title, int numPages) {
        this.title = title;
        this.numPages = numPages;
    }

    public void setTitle(String title) {
        this.title = title;
    }

    public void setNumPages(int numPages) {
        this.numPages = numPages;
    }

    public String getTitle() {
        return this.title;
    }

    public int getNumPages() {
        return this.numPages;
    }
}
```

Understanding what `this` really means is very simple: Conceptually, each object in Java has an implicit instance variable, named `this`, that points to the object itself. If you model a `Book` object as shown below, it is straightforward to understand what `this` is all about.

Suppose the object created is `new Book("Java", 345)`. Then, conceptually, this object is

![Diagram of a Book object](image)

**NOTE:** This is NOT what really happens during execution. In Java, each object does NOT keep a reference to itself – that would be a waste of memory space. However, take this as simply a conceptual model that helps us understand what `this` means.

**Using this to call constructors**

Inside a constructor, you may call another constructor, using `this()`, with properly given arguments. For a good example of this use, see the `Rational` class example.