

CHAPTER – 10

NUMBER MATTERS

static final constants

static final variables are constants

A variable marked **final** means that—once initialized—it can never change. In other words, the value of the static final variable will stay the same as long as the class is loaded. Look up `Math.PI` in the API, and you'll find:

```
public static final double PI = 3.141592653589793;
```

The variable is marked **public** so that any code can access it.

The variable is marked **static** so that you don't need an instance of class `Math` (which, remember, you're not allowed to create).

The variable is marked **final** because `PI` doesn't change (as far as Java is concerned).

There is no other way to designate a variable as a constant, but there is a naming convention that helps you to recognize one. ***Constant variable names are usually in all caps!***

A *static initializer* is a block of code that runs when a class is loaded, before any other code can use the class, so it's a great place to initialize a static final variable.

```
class ConstantInit1 {  
    final static int X;  
    static {  
        X = 42;  
    }  
}
```

Initialize a *final* static variable:

- ① At the time you declare it:

```
public class ConstantInit2 {  
    public static final int X_VALUE = 25;  
}
```

Notice the naming convention—static final variables are constants, so the name should be all uppercase, with an underscore separating the words.

OR

- ② In a static initializer:

```
public class ConstantInit3 {  
    public static final double VAL;
```

```
    static {  
        VAL = Math.random();  
    }  
}
```

This code runs as soon as the class is loaded, before any static method is called and even before any static variable can be used.

If you don't give a value to a final variable in one of those two places:

```
public class ConstantInit3 {  
    public static final double VAL;  
}
```

no initialization!

The compiler will catch it:

```
File Edit Window Help Init?  
% javac ConstantInit3.java  
ConstantInit3.java:2: error: variable VAL not initialized in the default constructor  
    public static final double VAL;  
                                ^  
1 error
```

final isn't just for static variables...

You can use the keyword **final** to modify non-static variables too, including instance variables, local variables, and even method parameters. In each case, it means the same thing: the value can't be changed. But you can also use final to stop someone from overriding a method or making a subclass.

non-static final variables

```
class Foof {
    final int size = 3;    ← now you can't change size
    final int whuffie;

    Foof() {
        whuffie = 42;    ← now you can't change whuffie
    }

    void doStuff(final int x) {
        // you can't change x
    }

    void doMore() {
        final int z = 7;
        // you can't change z
    }
}
```

in the result (formatted) string.

You've already seen some examples:

%,d means “insert commas and format the number as a decimal integer.”

and

%.2f means “format the number as a floating point with a precision of two decimal places.”

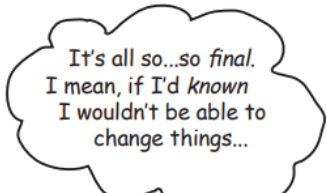
and

%,.2f means “insert commas and format the number as a floating point with a precision of two decimal places.”

A **final variable** means you can't change its value.

A **final method** means you can't override the method.

A **final class** means you can't extend the class (i.e., you can't make a subclass).



It's all so...so *final*.
I mean, if I'd *known*
I wouldn't be able to
change things...

EXERCISE

1.BE THE COMPILER

super static block

static block 3

in main

super constructor

constructor

REASON- **Static blocks** are executed when the class is first loaded, before the main method runs.**Parent class static blocks** are executed before child class static blocks.**Constructors** are executed when an instance of the class is created, following the order of inheritance.

2.TRUE OR FALSE

1.To use the Math class, the first step is to make an instance of it.

False

2.You can mark a constructor with the keyword “static.”

False

3.Static methods don't have access to an object's instance variables.

True

4.It is good practice to call a static method using a reference variable.

False

5.Static variables could be used to count the instances of a class.

True

6.Constructors are called before static variables are initialized.

False

7.MAX_SIZE would be a good name for a static final variable.

True.

8.A static initializer block runs before a class's constructor runs.

True

9.If a class is marked final, all of its methods must be marked final.

False

10.A final method can be overridden only if its class is extended.

False

11. There is no wrapper class for boolean primitives.

True

12. A wrapper is used when you want to treat a primitive like an object.

True

13. The parseXxx methods always return a String.

False

14. Formatting classes (which are decoupled from I/O) are in the java.format package.

False