

## Style

---

See also: Chapter 7

### Why is Style Important?

```
int operate(double d, vector<int> v) {  
    return (d * v[0]);  
}
```



What the heck is  
this code doing?

1. Your code may outlast you
2. Other people will be modifying your code
3. Self documented code helps you avoid time-consuming documentation and questions
4. In 3 months, even you won't know what you were thinking!

## Method and Function Comments

Use method/function comments to give usage instructions and explain things that can't be determined from the code:

```
/*  
 * adjustVolume()  
 *  
 * Sets the player volume based on the user's  
 * preferences  
 *  
 * This method will throw an "UninitializedPlayerException"  
 * if the initialize() method has not yet been called.  
 *  
 * Returns an int, indicating the new volume.  
 */
```

## Inline Comments

```
void sort(int inArray[], int inSize)
{
    for (int i = 1; i < inSize; i++) {
        int element = inArray[i];

        int j = i - 1;
        while (j >= 0 && inArray[j] > element) {
            inArray[j+1] = inArray[j];
            j--;
        }
        inArray[j+1] = element;
    }
}
```

```
/*
 * Implements the "insertion sort" algorithm.
 */
void sort(int inArray[], int inSize) {
    // Start at position 1 and examine each element.
    for (int i = 1; i < inSize; i++) {
        int element = inArray[i];

        // j marks the position in the sorted part
        int j = i - 1;
        // As long as the current slot in sorted is higher
        // than the element shift over & move backwards.
        while (j >= 0 && inArray[j] > element) {
            inArray[j+1] = inArray[j];
            j--;
        }
        // At this point the current position in the sorted
        // isn't greater than element, so this its pos.
        inArray[j+1] = element;
    }
}
```

## Metadata Comments

```
/*
 * Author: klep
 * Date: 040324
 * Feature: PRD version 3, Feature 5.10
 */
int adjustVolume()
{
    if (fUninitialized) {
        throw UninitializedPlayerException();
    }

    int newVol =
        getPlayer()->getOwner()->getPreferredVolume();

    if (newVol == -1)
        return -1; // Fix for bug #142 - jsmith 040330

    setVolume(newVol); // TODO: catch excpt? - akshayr 040401

    return newVol;
}
```

## Commenting Every Line – Too Much?

```
int result;          // Declare an integer to hold the result.
```

```
result = doodad.getResult(); // Get the doodad's result.
```

```
if (result % 2 == 0) { // If the result mod 2 is 0...  
    logError();        // then log an error,  
} else {              // otherwise . . .  
    logSuccess();     // log success.  
}                    // End if/else
```

```
return (result);     // Return the result
```

## Justifiable Commenting of Every Line:

```
// Call the calculate method with the default values.
result = doodad.calculate(getDefaultStart(),
getDefaultEnd(), getDefaultOffset());

// To determine success or failure, we need to bitwise AND
// the result with the processor-specific mask (see docs,
// page 201).
result = result & getProcessorMask();

// Set the user field based on the "Marigold Formula."
setUserField( (result + kMarigoldOffset) /
               MarigoldConstant) + MarigoldConstant );
```

## Fixed Format Commenting

```
// $Id: Watermelon.cpp,v 1.6 2004/03/10 12:52:33 klep Exp $
/**
 * Implements the basic functionality of a watermelon
 * TODO: Implement updated algorithms!
 */
class Watermelon {
public:
/**
 * @param initialSeeds The starting number of seeds
 */
Watermelon(int initialSeeds);

/**
 * Computes the seed ratio, using the Marigold
 * algorithm.
 *
 * @param slowCalc Whether or not to use long calcs
 * @return The marigold ratio
 */
double calcSeedRatio(bool slowCalc);
};
```

## Watermelon Class Reference

Implements the basic functionality of a watermelon. [More...](#)

```
#include <Watermelon.h>
```

[List of all members.](#)

### Public Member Functions

Watermelon (int initialSeeds)
double calcSeedRatio (bool slowCalc)
<i>Computes the seed ratio, using the Mangold algorithm.</i>

### Detailed Description

TODO: Implement updated algorithms!

Definition at line 6 of file [Watermelon.h](#).

### Constructor & Destructor Documentation

**Watermelon::Watermelon ( int *initialSeeds* )**

**Parameters:**

*initialSeeds* The starting number of seeds

### Member Function Documentation

**double Watermelon::calcSeedRatio ( bool *slowCalc* )**

**Parameters:**

*slowCalc* Whether or not to use long (slow) calculations

## Commenting Rules of Thumb

- Do your best to avoid offensive or derogatory language.
- Liberal use of inside jokes is generally considered okay.
- Reference bug numbers or feature IDs when possible.
- Include your initials and the date if you think somebody might want to follow up on the comment with you in the future.
- Resist the temptation to include somebody *else's* initials and the date to avoid having to take responsibility for the code.
- Remember to update your comments when you update the code.
- If you use comments to separate a function into sections, consider whether the function might be broken into multiple, smaller functions.

## Self-Documenting Code

```
void copyString(const char* inSource, char *outDest)
    int position = 0;

    while (inSource[position] != '\0') {
        outDest[position] = inSource[position];
        position++;
    }

    outDest[position] = '\0';
}
```

```
void copyString(const char* inSource, char* outDest)
{
    int i = 0;
    while (outDest[i] = inSource[i++]);
    outDest[i] = '\0';
}
```

# Decomposition

```
void someFunction(int arg1, char arg2, Structure *arg3)
{
    egypt_efeef(arg1, arg3);
    deff dfdf;
    int efefef;

    if (afa & feff aa arg2 || arg3->efef) {
        dfdf df = dfdf;
        arg3->wefefwef(df);
        cout << arg3;
        cerr << "wefef defdf d dfdf " << endl;
    } else {
        dfdfdf ad fdfdf;
        cout << arg3->wefefe;
        cerr << "werer" << endl;
    }

    // now do something else
    cout << "thing: " << arg3->efwefe << endl;
    cout << "thing2: " << arg3->wefef << endl;
    cout << "thing3: " << arg3->f3f3ef << endl;
}
```

```
void someFunction(int arg1, char arg2, Structure *arg3)
{
    egypt_efeef(arg1, arg3);

    if (adfdf()) {
        thing1();
    } else {
        thing2();
    }

    thing3();
}

bool adfdf()
{
    return afa & feff aa arg2 || arg3->efef;
}

void thing1()
{
    dfdf df = dfdf;
    arg3->wefefwef(df);
    cout << arg3;
    cerr << "wefef defdf d dfdf " << endl;
}

void thing2()
{
    dfdfdf ad fdfdf;
    cout << arg3->wefefe;
    cerr << "werer" << endl;
}

void thing3()
{
    cout << "thing: " << arg3->efwefe << endl;
    cout << "thing2: " << arg3->wefef << endl;
    cout << "thing3: " << arg3->f3f3ef << endl;
}
```

## Refactoring

The principle of *Refactoring* acknowledges that over time, code gets *cruffy* and needs to be reorganized.

- Refactoring keeps your code simple and readable.
- Refactoring is sometimes hard to justify.
- Unit tests help keep refactoring safe.
- In XP, you constantly refactor.

## Naming

<b>Good Names</b>	<b>Bad Names</b>
<code>srcName, dstName</code> Distinguishes two objects	<code>thing1, thing2</code> Too general
<code>gSettings</code> Conveys global status	<code>globalUserSpecificSettingsAndPreferences</code> Too long
<code>mNameCounter</code> Conveys data member status	<code>mNC</code> Too obscure, concise
<code>performCalculations()</code> Simple, accurate	<code>doAction()</code> Too general, imprecise
<code>mTypeString</code> Easy on the eyes	<code>_typeSTR256</code> A name only a computer could love
<code>Grid&lt;int&gt; inGridBergman</code> <code>return ofTheJedi;</code> Good jokes	<code>mIHateLarry</code> Inappropriate

Consistency is just as important as choosing a name.

## Klep's Personal Coding Conventions

mNumBlurbs	(data member)
sSchedule	(static)
kMenuFileOpen	(constant)
isBold	(boolean)
inBlurb	(input argument)
outNumBlurbs	(ref arg that will be modified)
CapitalizedName	(class)
lowerCaseName()	(method or function)
myBlurb	(object or variable)

```
class Foo
{
    public:
        bool myMethod(int inValue);
};

bool Foo::myMethod(int inValue)
{
    if (inValue > 7) {
        cout << "Greater than 7" << endl;
    } else {
        cout << "Not greater than 7" << endl;
    }

    return true;
}
```

Fortunately, some newer IDEs can automatically enforce style guidelines or convert between styles.