

## CSA Programming Style Guidelines

Every programming language has its own set of stylistic conventions that are meant to promote readability. The purpose of these conventions is to promote readability. While everyone agrees that improving readability of a program is desirable, everyone has a different idea on how to achieve that. For example, Acme Software Inc. may insist that their C++ programmers prefix all data members with “m\_” (e.g., m\_examScore, m\_name), while another company may not have this requirement. An Internet search on programming conventions will yield dozens of stylistic conventions, for dozens of different languages. Some of these documents will approach being one hundred pages long!

This document is not meant to be yet another all-encompassing document on coding conventions. Instead, it is a small set of ideas meant to guide beginning and intermediate level programmers in some of the more fundamental aspects of programming style. Inevitably, programmers encounter languages/situations that are not documented here, or worse yet, do not have established coding conventions. In these situations, you must use your past experiences to make good decisions. Although, there is not a concrete set of rules, you can generally rely on the following rules:

- Keep it simple
- Strive for clarity
- Be consistent
- Follow other well-established standards, if possible

Here are a few existing coding conventions for some common programming languages:

- **C++** – <http://www.chris-lott.org/resources/cstyle/indhill-cstyle.html>
- **Java** – <http://java.sun.com/docs/codeconv/html/CodeConvTOC.doc.html>
- **C#** – <http://www.csharpfriends.com/Articles/getArticle.aspx?articleID=336>
- **Visual Basic** – <http://www.gotdotnet.com/team/vb/VBSampleGuidelines.htm>

### Indentation, Whitespace, Wrapping

Indentation and whitespace should be used to improve the readability of your programs. This is a principle that holds in all programming languages. Indentation should be used to emphasize the nesting of control structures. Blank lines should be used to separate code sections that are performing logically different tasks. Lastly, lines longer than 70 characters should be wrapped onto the next line in a manner that enhances readability.

## Bracing

The bodies of conditionals and loops is often surrounded by braces, {}, or begin-end pairs. There are a number of acceptable styles. The example at the end shows one reasonable bracing style. A few acceptable approaches:

```
if (a == 0) {
    System.out.println("zero");
} else {
    System.out.println("non zero");
}
```

```
if (a == 0)
{
    System.out.println("zero");
}
else
{
    System.out.println("non zero");
}
```

Whatever style you adopt, it is important to be consistent throughout your program. Additionally, do not adopt a style that deemphasizes, or hides, the bracing:

```
if (a == 0) {
    System.out.println("zero"); }
else {
    System.out.println("non zero"); }
```

## Comments

Regardless of the language, every file should have the following information. When possible, you should use any conventions that allow the automatic generation of documentation (e.g., Java and C#).

- Your name
- Instructor name
- Course number and section
- Description of the file's contents
- Statement identifying code segments that were developed in full, or in part, by another person.

Each function or method will typically have the following information.

- Name of method
- Description of what method does
- Description of parameters
- Description of return value
- Assumptions
- Example usage, if it would help clarify how to use it

## Naming Conventions

There are three common conventions for naming programmatic entities. These are Pascal case, camel case, and uppercase. Pascal case capitalizes the initial letter of each word. Camel case is similar, but does not capitalize the first word. Uppercase uses all uppercase characters, but separates the words with an underscore. Here are some examples:

- Pascal Casing - TestCounter
- Camel Casing - testCounter
- Uppercase – TEST\_COUNTER

Each language has different conventions for casing. The following table identifies the most commonly used conventions.

	<b>C#</b>	<b>Java</b>	<b>C++</b>	<b>VB</b>
Class	Pascal	Pascal	Pascal	Pascal
Struct	Pascal	-----	Pascal	-----
Types	Pascal	-----	Pascal	-----
Interface	Pascal (prefix w/ "I")	Pascal	-----	Pascal
enum type	Pascal	Pascal	Pascal	Pascal
enum value	Pascal	Pascal	Pascal	Pascal
fields/data members	Camel	Camel	Camel	Pascal
Methods/property	Pascal	Camel	Pascal	Pascal
namespace/package	Pascal	Pascal	Pascal	Pascal
Constants	Upper	Upper	Upper	Pascal

## Example in Java

```
/**
 * John Doe
 * Instructor: Prof. Dolittle
 * CSA 174, Section B
 *
 * This main program provides an interface for a simple bank
 * account.
 *
 * All aspects of this program were developed by John Doe.
 */

import java.text.DecimalFormat;
import java.text.NumberFormat;

public class OneAccountATM {

    public static final int DISPLAY = 0;
    public static final int BALANCE = 1;
    public static final int WITHDRAWAL = 2;
    public static final int DEPOSIT = 3;
    public static final int PROJECTION = 4;
    public static final int QUIT = 5;

    public static void main(String[] args)
    {
        NumberFormat money = NumberFormat.getCurrencyInstance();
        DecimalFormat percent = new DecimalFormat("#0.0%");
        BankAccount account = new BankAccount();

        // Get input

        do {
            System.out.print( "Enter owner name: " );
        } while( !account.setName( ConsoleIn.readLine().trim() ) );

        do {
            System.out.print( "Enter owner age: " );
        } while( !account.setAge( ConsoleIn.readLineInt() ) );

        do {
            System.out.print( "Enter initial balance: " );
        } while( !account.setBalance( ConsoleIn.readLineDouble() ) );

        // Process series of user commands

        boolean doneFlag = false;

        do {

            System.out.println( "\nService Menu: " );
            System.out.println( "\n\tEnter code for the desired service." );
```

```

System.out.println( "\tAccount Summary:    \t0" );
System.out.println( "\tView Balance:      \t1" );
System.out.println( "\tMake withdraw:    \t2" );
System.out.println( "\tMake deposit:     \t3" );
System.out.println( "\tBalance Projection:\t4" );
System.out.println( "\tQuit:             \t5" );

System.out.print( "> " );
int code = ConsoleIn.readLineInt();

switch (code){
    case DISPLAY:
        System.out.println( account );
        break;
    case BALANCE:
        System.out.println( "Current Balance: " +
                            money.format( account.getBalance() ) );
        break;
    case WITHDRAWAL:
        System.out.print( "Enter amount to withdraw: ");
        account.withDraw( ConsoleIn.readLineDouble() );
        System.out.println( "New Balance: " +
                            money.format(account.getBalance()) );
        break;
    case DEPOSIT:
        System.out.print( "Enter amount to deposit: ");
        account.deposit( ConsoleIn.readLineDouble() );
        System.out.println( "New Balance: " +
                            money.format(account.getBalance()) );
        break;
    case PROJECTION:
        System.out.print( "Enter age for balance projection: ");
        int age = ConsoleIn.readLineInt();
        double bal = account.projectedBalance( age );
        if ( bal >= 0.0 ) {
            System.out.println( "Balance at age " + age +
                                " is projected to be " +
                                money.format( bal ) );
        } else {
            System.out.println( "Age is less that current age.");
        }
        break;
    case QUIT:
        doneFlag = true;
        break;
    default:
        System.out.println( "Illegal Action Code!" );
        break;
}
} while ( doneFlag != true );

System.out.println( "Thank you for using the one account ATM." );
}
}

```

```

/**
 * John Doe
 * Instructor: Prof. Dolittle
 * CSA 174, Section B
 *
 * This file implements a class definition for a simple bank
 * account.
 *
 * All aspects of this program were developed by John Doe.
 */

import java.text.DecimalFormat;
import java.text.NumberFormat;

public class BankAccount {

    public static final int MAX_NAME_LENGTH = 30;
    public static final int MAX_AGE = 125;
    public static final double INTEREST_RATE = 0.1;

    private double balance = 0.0;
    private int age = 0;
    private String ownerName = "no name";

    private static final NumberFormat money =
        NumberFormat.getCurrencyInstance();
    private static final DecimalFormat percent =
        new DecimalFormat("#0.0%");

    // ACCESSORS

    /**
     * @return account balance
     */
    public double getBalance() {
        return balance;
    }

    /**
     * @return owner's age.
     */
    public int getAge() {
        return age;
    }

    /**
     * @return owner's name.
     */
    public String getName() {
        return ownerName;
    }
}

```

```

/**
 * Determines if a name is legal.
 * @param name String representing a name
 * @return true if name is legal
 */
private static boolean nameOK(String name)
{
    boolean OK = true;
    int len = name.length();

    if (len < MAX_NAME_LENGTH) {
        for (int i=0; i<len; i++) {
            char ch = name.charAt(i);
            boolean isLetter = (ch >= 'A' && ch <= 'Z') ||
                               (ch >= 'a' && ch <= 'z');
            boolean isSpace = (ch == ' ');
            if (!isLetter && !isSpace) {
                OK = false;
            }
        }
    } else {
        OK = false;
    }

    return OK;
}

/**
 * Sets the account-owner's name. The name must be less than 29 characters
 * and contain only letters and spaces. If not, the name is left
 * unchanged.
 * @param name Owner's name
 * @return true if name is legal
 */
public boolean setName(String name) {
    if (nameOK(name)) {
        ownerName = name;
        return true;
    } else {
        return false;
    }
}

/**
 * Sets the account-owner's age. The age must be between 0 and 125.
 * If not, the age is left unchanged.
 * @param newAge Owner's age
 * @return true if age is legal
 */
public boolean setAge(int newAge) {
    if (newAge >= 0 && newAge <= MAX_AGE) {
        age = newAge;
        return true;
    } else {
        return false;
    }
}

```

```

/**
 * Sets the account's balance. The balance must not be negative.
 * @param amount New account balance.
 * @return true if balance is legal
 */
public boolean setBalance(double amount) {
    if (amount >= 0) {
        balance = amount;
        return true;
    } else {
        return false;
    }
}

/**
 * Adds money to account. The deposit amount must not be negative.
 * @param amount Amount to deposit.
 * @return true if deposit amount is legal
 */
public void deposit(double amount) {
    if (amount > 0.0) {
        balance += amount;
    } else {
        balance -= amount;
    }
}

/**
 * Takes money from account. The withdrawl amount must not be negative
 * and must not exceed the current balance.
 * @param amount Amount to withdrawl.
 * @return true if withdrawl succeeded
 */
public void withDraw(double amount) {
    if (amount >= 0.0 && amount <= balance) {
        balance -= amount;
    } else {
        balance += amount;
    }
}

/**
 * Determines how much the account will have when the owner
 * reaches a certain age. -1 is returned if the future age
 * is less than the current age.
 * @param futureAge Target age.
 * @return -1 if error condition, or projected balance at furureAge
 */
public double projectedBalance(int futureAge) {
    double futureBalance = balance;

    if (futureAge >= age) {
        for (int i = 0; i < futureAge - age; i++) {
            futureBalance += futureBalance * INTEREST_RATE;
        }
        return futureBalance;
    } else {

```

```
        return -1.0;
    }

}

/**
 * Convert account into String representation.
 * @return String representation of bank account
 */
public String toString() {
    return "Bank Account Summary\n"
        + "\tOwner Name: "
        + ownerName
        + "\n\tOwner Age: "
        + age
        + "\n\tInterest Rate: "
        + percent.format(INTEREST_RATE)
        + "\n\tBalance: "
        + money.format(balance);
}

}
```