Module1: Coding Standards
Software Development

Fall 2023

Dr. William Greg Johnson
Department of Computer Science
Georgia State University

Georgia State University | COLLEGE OF ARTS & SCIENCES

Module1: Coding Standards

- Fundamental Principles
  - Why important?
  - Guidelines, not templated formats
  - Leverages syntax of program to help understand ("My code is self documented…")
  - Style of variable names, indentation, comments add value to your future self and others (Code conventions)

GeorgiaState University | COLLEGE OF ARTS & SCIENCES

Module1: Coding Standards

First, start with:

- **Programming Principles**
  - The main goal of the programmer is to make simple and easy to read programs with few bugs in it
  - How?
    - Structured programming
    - Information hiding
    - Programming practices (code review, style checker)
    - **Coding standards**

Coding Standard Principles

- Maintainability is the ease with which a product can be modified in order for:
  - correcting defects or their cause
  - meeting new requirements
  - coping with a changing environment
- Reduce the cost of software maintenance
  - Approximately 60%-80% of the lifetime cost of a software system goes to maintenance
- Code conventions are not enforced by any compiler or IDE –yet…
  - Future IDE with an AI engine for code convention processing

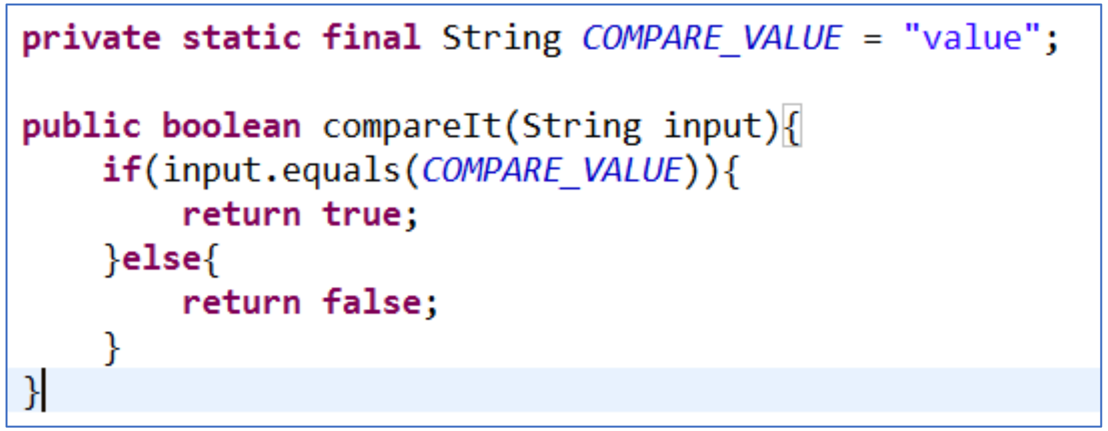
GeorgiaState University | COLLEGE OF ARTS & SCIENCES

Coding Standards, Why?

- Most of the time software is not maintained by the original author
- If everyone follows the standards, you feel comfortable using your colleague's code, because it seems like your own code
- Not following coding standards will lead to review comments and a refactor, leading to effort variation
- It’s a vicious cycle. The only way out is following coding standards.

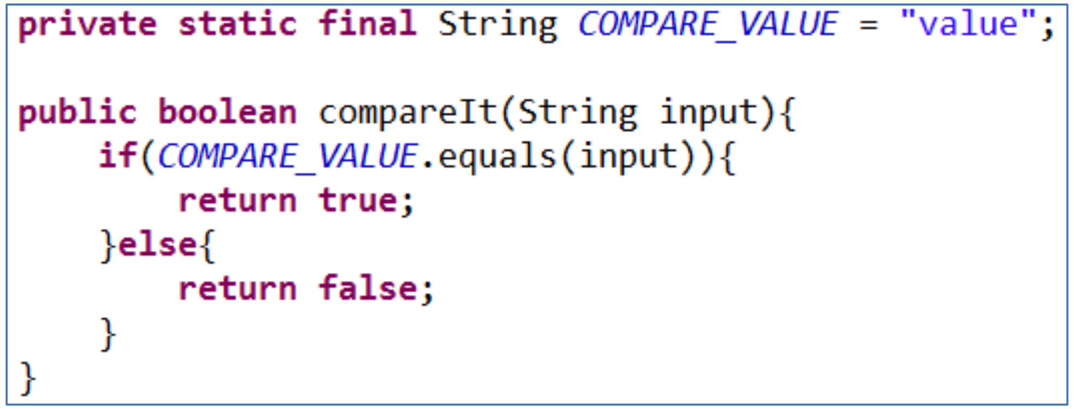
Coding Standard Practices

- Constants, use on left hand side.
  - Avoids invalid assignment
  - Avoids null pointer exceptions



```java
private static final String COMPARE_VALUE = "value";

public boolean compareIt(String input){
    if(input.equals(COMPARE_VALUE)){
        return true;
    }else{
        return false;
    }
}
```

bad practice, this can cause null pointer exception if parameter ’input’ is null



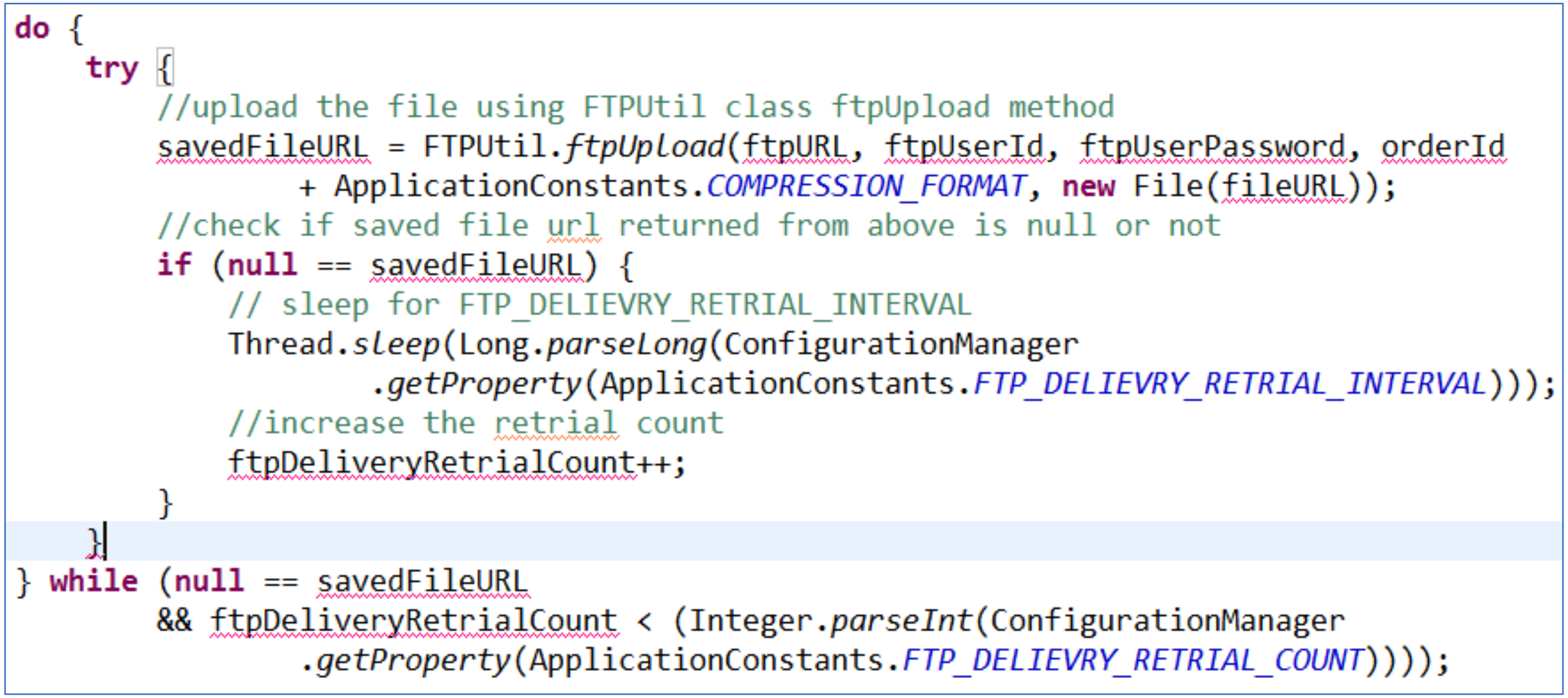
```java
private static final String COMPARE_VALUE = "value";

public boolean compareIt(String input){
    if(COMPARE_VALUE.equals(input)){
        return true;
    }else{
        return false;
    }
}
```

good practice

Coding Standard Practices

- Indentation
  - gives visual indicator of scope (Warning: Python is space sensitive)
  - gives better readability of code
  - gives maintainable code for future software developers
- Commenting
  - clearly describe the functions of your code for ‘future’ self and others
  - should be complete and succinct

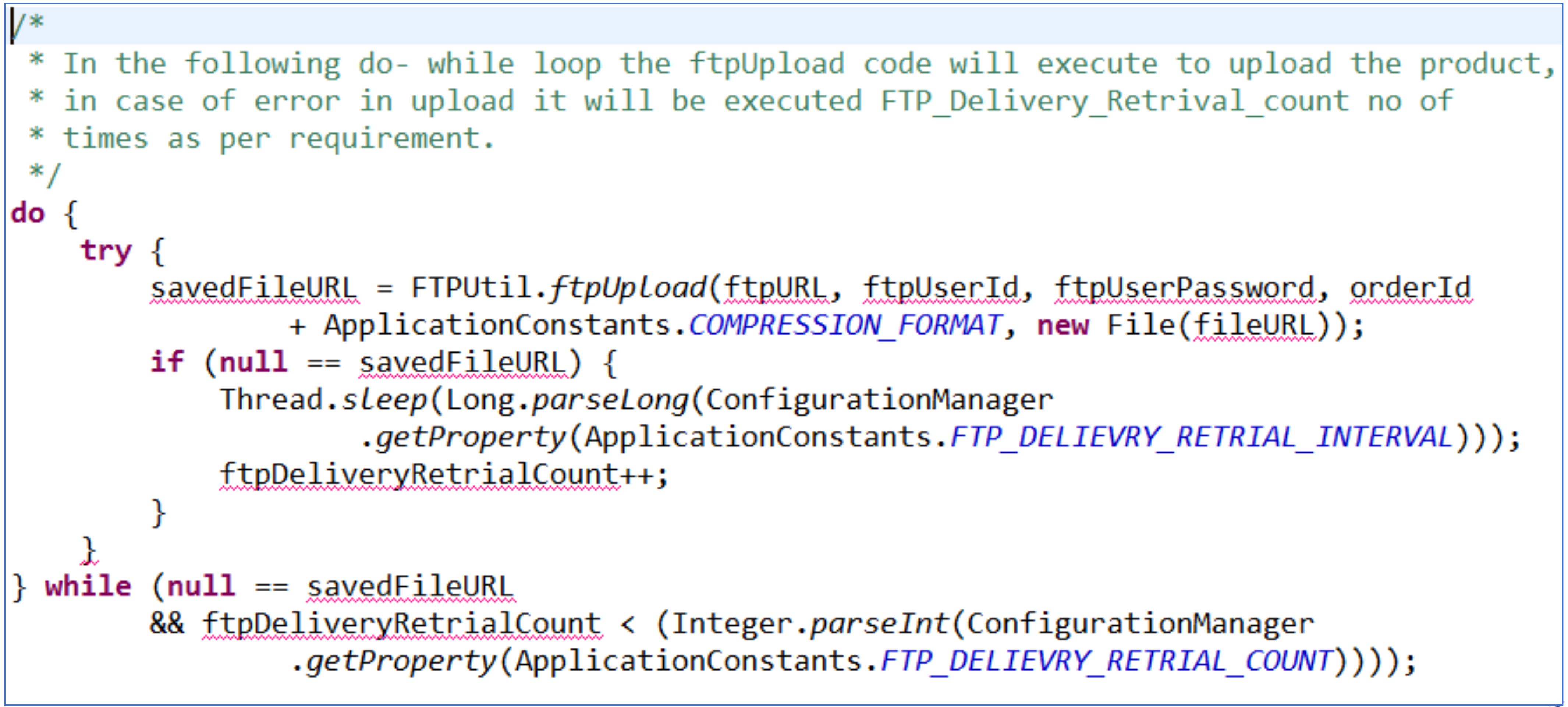
Coding Standards with comments



```
do {
    try {
        //upload the file using FTPUtil class ftpUpload method
        savedFileURL = FTPUtil.ftpUpload(ftpURL, ftpUserId, ftpUserPassword, orderId
                + ApplicationConstants.COMPRESSION_FORMAT, new File(fileURL));
        //check if saved file url returned from above is null or not
        if (null == savedFileURL) {
            // sleep for FTP_DELIEVRY_RETRIAL_INTERVAL
            Thread.sleep(Long.parseLong(ConfigurationManager
                    .getProperty(ApplicationConstants.FTP_DELIEVRY_RETRIAL_INTERVAL)));
            //increase the retrial count
            ftpDeliveryRetrialCount++;
        }
    }
} while (null == savedFileURL
        && ftpDeliveryRetrialCount < (Integer.parseInt(ConfigurationManager
                .getProperty(ApplicationConstants.FTP_DELIEVRY_RETRIAL_COUNT))));
```

bad practice, with imbedded, unhelpful comments

Coding Standards with comments



```
/*
 * In the following do- while loop the ftpUpload code will execute to upload the product,
 * in case of error in upload it will be executed FTP_Delivery_Retrival_count no of
 * times as per requirement.
 */
do {
    try {
        savedFileURL = FTPUtil.ftpUpload(ftpURL, ftpUserId, ftpUserPassword, orderId
                + ApplicationConstants.COMPRESSION_FORMAT, new File(fileURL));
        if (null == savedFileURL) {
            Thread.sleep(Long.parseLong(ConfigurationManager
                    .getProperty(ApplicationConstants.FTP_DELIEVRY_RETRIAL_INTERVAL)));
            ftpDeliveryRetrialCount++;
        }
    }
} while (null == savedFileURL
        && ftpDeliveryRetrialCount < (Integer.parseInt(ConfigurationManager
                .getProperty(ApplicationConstants.FTP_DELIEVRY_RETRIAL_COUNT))));
```

- Comments are not about how much you write
- Comments should add illustration to the code for maintainability

Coding Standards for naming variables

- Names are considered less important, but are essential to making sense of code segments
- In the most widely used programming languages (Python, Java, JavaScript), variables can be almost anything
- Pascal Casing: first character of word is Upper Case and others are lower
  - Example: BlueColor
- Camel Casing: first character of all words except the first one are Upper Case and all other are lower
  - Example: blueColor

Coding Standards for naming

1. Use Pascal casing for method names:

```
void SayHello( String name )
{
      ...
}
```

2. Use Camel casing for variables and method parameters:

```
int totalCount = 0;
void SayHello( String pName )
{
      String fullMessage = "Hello " + pName;
      ...
}
```

Coding Standards for naming

3. Use the prefix '*I*' with Camel casing for interfaces (Example: *IOrderingForm*)
4. Do not use Hungarian notation to name variables.
   - In earlier days most of the programmers liked having the data type as a prefix for the variable name and using *m_* as prefix for member variables.
     - E.G., string *m_sName; int nAge;*
     - All variables should use Camel casing.
   - Some programmers still prefer to use the prefix *m_* to represent member variables, since there is no other easy way to identify a member variable

Coding Standards for naming

5. Use Meaningful, descriptive words to name variables. Do not use abbreviations.

Good:	Bad:
`String address;`	`String addr;`
`int salary;`	`int sal;`

6. Do not use single character variable names, e.g., **`i`**, **`n`**, **`s`**... Use meaningful names, e.g., index, temp. One exception in this case would be variables used for iterations in loops, e.g.:

```
for ( int i = 0; i < count; i++ )
{
    ...
}
```

If the iterator variable is used only for iteration, and is not used elsewhere in the loop body, it is acceptable to use a single char, e.g., **`i`**, **`j`**, **`k`**...

Coding Standards for naming

7. Do not use underscore (what?) for local variable names.
8. Use underscore `‘_’` to prefix all member variables to distinguish from local variables.
9. Avoid variable names close to syntax of the programming language, e.g., `Scanner scanner = new Scanner(System.in);`
10. Prefix Boolean variables, properties, methods with `‘is’` or similar, e.g., `private boolean _isFinished;`
11. Namespace identifiers need to adopt standard pattern:
    <company>.<product>.<top level>.<bottom level> (hint: you already use this…)
    E.G., `import java.util.ArrayList;`

Coding Standards for naming

12. Use appropriate prefix for the UX elements so that you can identify them from the rest of the variables.

    2 different approaches recommended here.

    - Use a common prefix (**`ux_`** or **`ui_`**) for all interface elements. This will help group elements together.
    - Use appropriate prefix indicating the component. (Below are a few examples)

13. Use Pascal case for file names. (Java mandates file name matches class name.)

Control	Prefix
Label	**`lbl`**
TextBox	**`txt`**
Button	**`btn`**
ListBox	**`lst`**
CheckBox	**`chk`**
RadioButton	**`rdo`**
Image	**`img`**

Coding Standards for indentation/spacing

1. Use TAB instead of spaces for indentation. (Set TAB to 4 spaces in IDE)
2. Comments move to same level as the segment it documents (code).

Good

```
    // Format a message to show hello message
    String fullMessage = "Hello " + firstName;
    JOptionPane.showMessageDialog( NULL, fullMessage );
    …
```

Bad

```
           //   Format a thing to show something
    String fullMessage = "Hello " +
      firstName;
           JOptionPane.showMessageDialog( NULL, fullMessage );
    …
```

Coding Standards for indentation/spacing

3. Curly braces '**{ }**' should be at the same level as a block of code they designate.

Good

```
// Method saying hello message
public void SayHello( String message )
{
    String firstName = "William";
    System.out.println( message + firstName );
    …
}
```

Bad

```
           //   Hello method doing some output
public void SayHello(String message)      {
String firstName= "William";
 System.out.println( message+firstName );
… }
```

Coding Standards for indentation/spacing

4. Curly braces **‘{ }’** should be on a separate line from statement starting a block of code. (Shown before.)
5. Use a single space before and after operands and operators

Good

```
    if( showResult >= 4 )
    {
        System.out.println( “Exceeded limit.” );
        …
    }
```

Bad

```
    if(showResult>=4)  {
        for(int   i =0;i<10 ; i++) {
                …
        }
    …
    }
```

Georgia State University | COLLEGE OF ARTS & SCIENCES

Coding Standards for indentation/spacing

6. If programming supports `#region` / `#EndRegion` then use to enable collapsible sections of code. (Some IDE programs support similar for Java.)
7. Keep private member variables, properties, methods at top of class and public ones at the bottom.

Good Programming Standard Practices

1. Avoid writing very long methods; typically, 1 to 25 lines of code. If it has more than 25 lines of code, consider refactoring into separate methods.
2. Method name should tell what it does. Do not use misleading names. If the method name is obvious, there is no need of documentation explaining what the method does.
3. A method should do only 'one job'. Do not combine several into a single method, even if those jobs are very small. This supports high cohesiveness, a very good practice in software development.

Good Programming Standard Practices

4. Avoid writing very long methods; typically, 1 to 25 lines of code. If it has more than 25 lines of code, consider refactoring into separate methods.
5. Always watch for unexpected values in ‘if’ testing. For example, if you are using a parameter with an assumed set of possible values, never assume that if one does not match, then the only possibility is one of the other values.

Good

```
If ( memberType == eMember.REGISTERED )
{
    // Registered user... do something...
}
else if ( memberType == eMember.GUEST )
{
    // Guest user... do something...
}
else
{
    // Un expected user type: exception
    throw new Exception( … );
}
```

Bad

```
If( memberType == eMemberTypes.Registered )
{
    // Registered user... do something...
}
else
{
    // Assumed a Guest user... do something...
    // If we introduce another user type in future,
    // fail and will not be noticed.
}
```

GeorgiaState University | COLLEGE OF ARTS & SCIENCES

Good Programming Standard Practices

6. Do not hardcode numbers. Use constants instead. Declare constant in the top of the file and use it in your code. Even better, use a configuration file or store them in a database so change is easier and no recompiling the source files.
7. Convert strings to lowercase or upper case before comparing. This will ensure the string will match even if the string being compared has a different case.
8. Use `String.length() == 0` or `String.isEmpty()` instead of ""

Good

```
If ( lastName.isEmpty() || lastName.length() == 0 )
{
    // do something
}
```

Good Programming Standard Practices

9. Avoid using member variables shared between methods. Declare local variables wherever necessary in a method and pass it to other methods. It will be difficult to track which method changed the value and when if declared outside and used in multiple places.
10. Use `enum` wherever required. Do not use numbers or strings to indicate discrete values.
11. Do not make the member variables public or protected. Keep them private and allow access/changing with public/protected methods.
12. If the required configuration file is not found, application should be able to create one with default values.

GeorgiaState University | COLLEGE OF ARTS & SCIENCES

Good Programming Standard Practices

13. Error messages should help the user to solve the problem. Never give error messages like "Error in Application", "There is an error" etc. Instead give specific messages like "Failed to update database. Please make sure the login id and password are correct.”
14. Show short and friendly message to the user. But log the actual error with all possible information. Using `try/catch` with logging is an industry standard.
15. Do not have more than one class in a single file. Avoid passing too many parameters to a method. If you have more than 4 to 5 parameters, it is a good candidate to define an interface or other custom data structure.

Good Programming Standard Practices

16. If you have a method returning a collection, return an empty collection instead of null, if you have no data to return. For example, if you have a method returning an ArrayList, always return a valid ArrayList. If you have no items to return, then return a valid ArrayList with 0 items. This will make it easy for the calling application to just check for the count rather than doing an additional check for null.
17. If you are opening database connections, sockets, or file streams, always close them in the `finally` block. This ensures if any exceptions occur after opening the connection, it will be safely closed in the finally block.

Good Programming Standard Practices

18. Use `try-catch` in your data layer to reveal all database exceptions. Log the details with the name of the command being executed, stored process name, parameters, connection string used, other information. After logging the exception, you could throw a custom exception so that another code layer can catch it and take appropriate action.

Good Programming Standard Practices: Recap

1. The purpose is any developer familiar with guidelines can work on any code that followed and used them.
2. Using coding standards saves both time and money.
3. Standardize early - the effort to bring your old work into the standard will be too great otherwise.
4. Document every time you violate a standard.
5. Industry Standards > organizational standards > project standards > no standards.

Attributions

1. Ankur Goyal, https://www.slideshare.net/ankur_slideshare/software-development-best-practices-coding-guidelines
2. Mimoh Ojha, https://www.slideshare.net/mimohojha/coding-standards-33045251