**GUIDE.03: INSTRUCTIONS FOR AVOIDING WHEN PROGRAM**

# I. Purpose

Provide detailed requirements and instructions on programming work for the construction, upgrading and deployment of Information Technology application systems in the Military Telecommunications Group to ensure the avoidance of potential errors. during application operation.

# II. Subjects, scope of application

This guide applies at all units and software centers that have software application development activities.

This guide applies to all applications developed using the Java programming language, . Net , Java Script/Html, PHP. Applications written in these programming languages need to make sure to handle all the source code requirements in the regulations before being officially deployed to end users.

# III. List of requirements for source code using the java programming language

*(Details on how to do this can be found in Appendix 01)*

|  |  |
| --- | --- |
| **STT** | **Request name** |
| first | Do not use "Double.longBitsToDouble" with a parameter of type "int" |
| 2 | Don't use "Lock" in "synchronized" block |
| 3 | Override two methods "equals(Object obj)" and "hashCode()" at the same time |
| 4 | Check the input parameter to the method "equals(Object obj)" |
| 5 | Don't put "return" in "finally" blocks |
| 6 | In a synchronized block on an object, no other value is assigned to that object. |
| 7 | Don't call "wait(...)", "notify()", "notifyAll()" method in Thread |
| 8 | Check for Null . value |
| 9 | Close resources after use |
| ten | Logical condition should be guaranteed to not access Null . object |
| 11 | The "Cloneables" interface implementation needs to override the "clone" method. |
| twelfth | If you override the "equals(Object obj)" or "compareTo(T obj)" method, override both methods. |
| 13 | The variable in the for loop stop condition and the variable that changes after each loop must be the same variable |
| 14 | Variables in the for loop stop condition should not be changed within the for loop or depend on results returned from other code. |
| 15 | Declare final with "public static" variables |
| 16 | Have the keyword "case" in each block of the switch declaration |
| 17 | Do not pass a Collection object to its own method |
| 18 | Do not throw exception in Servlet |
| 19 | Garbage cleanup is only done from the JVM |
| 20 | Does not compare equals with data type Float |
| 21 | No access to static properties from Instance's methods |
| 22 | Initialize a static field set in "synchronized" |
| 23 | Unleash the Lock |
| 24 | Don't declare "public static" with mutable fields |
| 25 | Do not use "Math.abs" for a possible value of "MIN\_VALUE" |
| 26 | Do not write non-serializable classes |
| 27 | Eliminate test conditions that never occur in if/else. |
| 28 | The return value of the function should not be ignored. |
| 29 | Do not use operators other than short-circuit| in conditional expressions |
| 30 | End Switch cases with "break" command |
| thirty first | Do not call Thread.run() and Runnable.run() directly |
| 32 | Don't call Throwable.printStackTrace(...). |
| 33 | When handling exception, it is necessary to preserve the original exception |
| 34 | Conditional expressions should not always be "TRUE" or "FALSE" |
| 35 | Do not declare static variables with objects of type "Calendars" and "DateFormats" |
| 36 | Don't compare Class type by Class Name |
| 37 | Do not use the same operator on both sides of a binary expression |
| 38 | Removed "dead stores" |
| 39 | When testing for equality with a String variable, Strings should be placed to the left of the equal . comparison expression |
| 40 | Use ConcurrentHashMap instead of HashMap |
| 41 | Use pool when calling webservice |
| 42 | Ensure the performance of updating data to redis using jedis . library |
| 43 | Note when using synchronized |

# I V. List of requirements for source code using the .NET programming language

*(Details on how to do this can be found in Appendix 02)*

|  |  |
| --- | --- |
| **STT** | **Request name** |
| first | Do not return the "IDisposables" object initialized in the code using "using" |
| 2 | Use short-circuit logic to avoid null memory access errors in comparison conditions |
| 3 | Objects that implement the "IDisposables" interface need to be disposed |
| 4 | The increment variable in the "for" loop must be the one in the loop stop condition |
| 5 | Anonymous Delegate cannot be used to unsubscribe from an event. |
| 6 | The class must release member variables in that Class's Dispose function. |
| 7 | If a class has a member variable of "IDisposable", that class should implement the "IDisposable" interface. |
| 8 | Do not log to Console |
| 9 | Don't virtualize Field-like events |
| ten | Do not compare unassigned generic type parameter with null |
| 11 | Expressions that are always true should not be used |
| twelfth | It is not recommended to use "OrderBy" consecutively in LinQ |
| 13 | The "if/else if" structure should not have the same conditions |
| 14 | Do not use Static fields in objects of generic type |

# V . List of requirements for source code using the JavaScript programming language, Html

*(Details on how to do this can be found in Appendix 03)*

|  |  |
| --- | --- |
| **STT** | **Request name** |
| first | Do not use "NaN" in comparison operations |
| 2 | Use short-circuit logic to avoid null memory access errors in comparison conditions |
| 3 | Do not use the "delete" operator with type Array |
| 4 | Do not manipulate data with "eval" and "arguments" variables |
| 5 | The increment variable in the "for" loop must be the one in the loop stop condition |
| 6 | Control when sending messages via text |
| 7 | Remove the "debugger" command before deploying |
| 8 | Don't use obvious expressions that are always true or always false |
| 9 | Do not use the same property name when declaring an object |
| ten | Conditions in "if/else if" or "switch…case" blocks do not overlap |
| 11 | Use the "length" attribute to specify DOM elements |
| twelfth | Don't use the global "this" variable |
| 13 | Model attribute names without spaces " " |
| 14 | Operators "+" and "-" are not used with Object |
| 15 | In the "password" input tag set the "autocomplete" attribute to "off" |

# V I. List of requirements for source code using the PHP programming language

*(Details on how to do this can be found in Appendix 04)*

|  |  |
| --- | --- |
| **STT** | **Request name** |
| first | Don't use $this in a static context |
| 2 | Do not use exit(...), die(...) |
| 3 | Do not define Functions, Variables outside of Class |
| 4 | Don't use the variable Variable variables |
| 5 | No hard-coded username/password |
| 6 | Don't use "global" |
| 7 | No dynamic code execution |
| 8 | Do not use the same operator on both sides of a binary expression |
| 9 | Do not declare other statements after Jump |
| ten | Do not create redundant objects |
| 11 | Do not use repeated conditions in the same expression "if/else if" and "cases" in |
| twelfth | Logical condition should be guaranteed to not access Null . object |
| 13 | Use break at the end of Switch cases . expressions |

# APPENDIX 01: PROGRAMMING GUIDELINES TO AVOID POTENTIAL ERRORS JAVA APPLY SOURCE CODE

#### Do not use "Double.longBitsToDouble" with a parameter of type "int"

Double.longBitsToDouble requires a 64bit long argument, so converting small numbers like int to double can cause errors due to incorrect bit placement.

Non-compliant code example

|  |
| --- |
| int i **=** 42 **;**  double d **=** Double **.** longBitsToDouble **(** i **);** // not compliant because i is of type int |

Correct spelling:

|  |
| --- |
| long i **=** 42 **;**  double d **=** Double **.** longBitsToDouble **(** i **);** |

#### Don't use "Lock" in the "synchronized ." block

java.util.concurrent.locks provides more flexible and powerful locking methods than synchronized block, using synchronize with Lock object will lose this advantage

Non-compliant code example

|  |
| --- |
| Lock lock **=** **new** MyLockImpl **();**  synchronized **(** lock **)** **{** // not synchronized with object of type Lock  //...  **}** |

Correct spelling:

|  |
| --- |
| Lock lock **=** **new** MyLockImpl **();**  lock **.** tryLock **();**  //... |

#### Override two methods "equals(Object obj)" and "hashCode()" at the same time

According to the Java language specification, there is a binding between the equals(Object) and hashCode() methods:

* + If two objects are equal according to the equals(Object) method, then calling the hashCode method with each object must return the same integer result.
  + If the two objects are not equal according to the equals(Object) method, then hashCode will return the results as distinct integers.

According to this constraint, the two methods should be overridden when used

Non-compliant code example

|  |
| --- |
| class MyClass **{** //not override "hashCode()"  @Override  public boolean equals **(** Object obj **)** **{**  /\* ... \*/  **}**  **}** |

Correct spelling:

|  |
| --- |
| class MyClass **{**  @Override  public boolean equals **(** Object obj **)** **{**  /\* ... \*/  **}**  @Override  public int hashCode **(** Object obj **)** **{**  /\* ... \*/  **}**  **}** |

#### K Checks the input parameter for the method "equals(Object obj)"

The "equals" method uses Object as an input parameter so any object can be passed in for comparison, it should not be assumed that only objects of the same type are passed, but need to be checked to make sure. no error occurred.

Non-compliant code example

|  |
| --- |
| public boolean equals **(** Object obj **)** **{**  MyClass mc **=** **(** MyClass **)** obj **;** // the passed obj object can be null or of other type MyClass  // ...  **}** |

Correct spelling:

|  |
| --- |
| public boolean equals **(** Object obj **)** **{**  **if** **(** obj **==** **null )**  **return** **false ;**  **if** **( this .** getClass **()** **!=** obj **.** getClass **())**  **return** **false ;**  MyClass mc **=** **(** MyClass **)** obj **;**  // ...  **}** |

#### Don't put "return" in "finally" blocks

Do not call return in finally block when handling exception, otherwise exceptions will not be thrown in try or catch() blocks if any

Non-compliant code example

|  |
| --- |
| public static void main **(** String **[]** args **)** **{**  **try** **{**  doSomethingWhichThrowsException **();**  System **.** out **.** println **(** "OK" **);** // this message is still displayed even though the print statement is not logically executed because the above method throws execution and forwards it into the catch block  **}** **catch** **(** RuntimeException e **)** **{**  System **.** out **.** println **(** "ERROR" **);** // this message will not be displayed  **}**  **}**  public static void doSomethingWhichThrowsException **()** **{**  **try** **{**  **throw** **new** RuntimeException **();**  **}** **finally** **{**  /\* ... \*/  **return ;** // declaring return here will cause the throw in the above try block to not be executed  **}**  **}** |

Correct spelling:

|  |
| --- |
| public static void main **(** String **[]** args **)** **{**  **try** **{**  doSomethingWhichThrowsException **();**  System **.** out **.** println **(** "OK" **);**  **}** **catch** **(** RuntimeException e **)** **{**  System **.** out **.** println **(** "ERROR" **);** // "ERROR" is printed as expected  **}**  **}**  public static void doSomethingWhichThrowsException **()** **{**  **try** **{**  **throw** **new** RuntimeException **();**  **}** **finally** **{**  /\* ... \*/  **}**  **}** |

#### In a synchronized block on an object, no other value is assigned to that object.

Synchronized on an object is essentially synchoronized on an instance (object instance) assigned to the object. Assigning a different value to that object in a synchoronized block will make this block possible to be run by other threads.

Non-compliant code example

|  |
| --- |
| private String color **=** "red" **;**  private void doSomething **(){**  synchronized **(** color **)** **{** // lock is actually performed on "red" referenced by the variable color  //...  color **=** "green" **;** //error, after performing the assignment other threads will be allowed to run this synchronized block  // ...  **}**  **}** |

Correct spelling:

|  |
| --- |
| private String color **=** "red" **;**  private Object lockObj **=** **new** Object **();**  private void doSomething **(){**  synchronized **(** lockObj **)** **{**  //...  color **=** "green" **;**  // ...  **}**  **}** |

#### Don't call "wait(...)", "notify()", "notifyAll()" method in Thread

Do not call these methods when using Thread because JVM relies on these methods to change Thread's state (BLOCKED, WAITING,..), calling them will mess up JVM's behavior.

Non-compliant code example

|  |
| --- |
| Thread myThread **=** **new** Thread **( new** RunnableJob **());**  **...**  myThread **.** wait **(** 2000 **);** |

#### Check for Null . value

When accessing the content in the NULL object, the program will appear a NullPointerException error, the program may encounter a business error or be stopped halfway, worse, hackers can take advantage of it to attack the system. Need to perform non-Null object check before accessing.

Notice we can use annotations: @CheckForNull and @Nonnull to indicate Null or non-Null values.

@Nullable pointed out that in some cases it is possible to take a Null value.

Non-compliant code example

|  |
| --- |
| @CheckForNull  String getName **()** **{...}**  public boolean isNameEmpty **()** **{**  **return** getName **().** length **()** **==** 0 **;** // Unchecked getName() function value other than Null  **}**  Connection conn **=** **null ;**  Statement stmt **=** **null ;**  **try** **{**  conn **=** DriverManager **.** getConnection **(** DB\_URL **,** USER **,** PASS **);**  stmt **=** conn **.** createStatement **();**  // ...  **}** **catch (** Exception e **)** **{**  e **.** printStackTrace **();**  **}** **finally** **{**  stmt **.** close **();** // untested stmt other Null  conn **.** close **();** // other unchecked conn Null  **}**  private void merge **(** @Nonnull Color firstColor **,** @Nonnull Color secondColor **){...}**  public void append **(** @CheckForNull Color color **)** **{**  merge **(** currentColor **,** color **);** // color should be checked other than Null because merge function doesn't accept Null value  **}**  void paint **(** Color color **)** **{**  **if (** color **==** **null )** **{**  System **.** out **.** println **(** "Unable to apply color " **+** color **.** toString **());** // color is not checked other Null  **return ;**  **}**  **...**  **}** |

#### Close resources after use

After use, it needs to be closed to free up resources for other processes to use and free up memory.

Non-compliant code example

|  |
| --- |
| OutputStream stream **=** **null ;**  **try {**  **for** **(** String property **:** propertyList **)** **{**  stream **=** **new** FileOutputStream **(** "myfile.txt" **);** //Open multiple streams  // ...  **}**  **} catch (** Exception e **){**  // ...  **} finally {**  stream **.** close **();** //Many streams are opened, but only the last one is closed  **}** |

Correct spelling:

|  |
| --- |
| OutputStream stream **=** **null ;**  **try {**  stream **=** **new** FileOutputStream **(** "myfile.txt" **);**  **for** **(** String property **:** propertyList **)** **{**  // ...  **}**  **} catch (** Exception e **){**  // ...  **} finally {**  stream **.** close **();**  **}**  //Java 7 introduces a way of writing code that closes all objects in the try statement as follows:  **try** **(** BufferedReader br **=** **new** BufferedReader **( new** FileReader **(** fileName **)))** **{**  //...  **}**  **catch** **(** **...** **)** **{**  //...  **}** |

#### Logical condition should be guaranteed to not access Null . object

When writing logical conditions, we need to pay attention to which condition is executed first, which condition is executed later to ensure that there are no cases of accessing the Null object.

Non-compliant code example

|  |
| --- |
| **if** **(** str **==** **null** **&&** str **.** length **()** **==** 0 **)** **{**  System **.** out **.** println **(** "String is empty" **);** //both conditions str == null and str.length() == 0 are executed, then if str is null then str.length() will cause an error  **}**  **if** **(** str **!=** **null** **||** str **.** length **()** **>** 0 **)** **{**  System **.** out **.** println **(** "String is not empty" **);**  **}** |

Correct spelling:

|  |
| --- |
| **if** **(** str **==** **null** **||** str **.** length **()** **==** 0 **)** **{**  System **.** out **.** println **(** "String is empty" **);** //if str is null then the condition str.length() will not be executed and will not raise an error  **}**  **if** **(** str **!=** **null** **&&** str **.** length **()** **>** 0 **)** **{**  System **.** out **.** println **(** "String is not empty" **);**  **}** |

#### The "Cloneables" interface implementation needs to override the "clone ." method

Implement Cloneables interface, it is necessary to override clone method, otherwise the default JVM clone method will be used then only primitive properties are copied, for other properties only reference copy (reference) ) to the clone object, the cloned object may share properties with the source object.

Non-compliant code example

|  |
| --- |
| class Team **implements** Cloneable **{** // Do not override clone()  private Person coach **;**  private List **<** Person **>** players **;**  public void addPlayer **(** Person p **)** **{...}**  public Person getCoach **()** **{...}**  **}** |

Correct spelling:

|  |
| --- |
| class Team **implements** Cloneable **{**  private Person coach **;**  private List **<** Person **>** players **;**  public void addPlayer **(** Person p **)** **{** **...** **}**  public Person getCoach **()** **{** **...** **}**  @Override  public Object clone **()** **{**  Team clone **=** **(** Team **)** **super .** clone **();**  //...  **}**  **}** |

#### If you override the "equals(Object obj)" or "compareTo(T obj)" method, override both methods.

According to the Java documentation on the Comparable.compareTo(T o) method: it is recommended but not required to guarantee that (x.compareTo(y)==0) is equivalent to (x.equals(y)).

So to avoid confusion later, if you override equals() or compareTo() method, you should override both methods together.

Non-compliant code example

|  |
| --- |
| public class Foo **implements** Comparable **<** Foo **>** **{**  @Override  public int compareTo **(** Foo foo **)** **{** /\* ... \*/ **}** // method equals(Object obj) must not be overridden  **}** |

Correct spelling:

|  |
| --- |
| public class Foo **implements** Comparable **<** Foo **>** **{**  @Override  public int compareTo **(** Foo foo **)** **{** /\* ... \*/ **}**  @Override  public boolean equals **(** Object obj **)** **{** /\* ... \*/ **}**  **}** |

#### The variable in the for loop stop condition and the variable that changes after each loop must be the same variable

The variable in the for loop stop condition and the variable that changes after each loop must be the same variable, otherwise the loop may never end.

Non-compliant code example

|  |
| --- |
| **for** **(** i **=** 0 **;** i **<** 10 **;** j **++)** **{** // condition to stop the loop based on the variable i, however i is not incremented in the last part of for  // ...  **}** |

Correct spelling:

|  |
| --- |
| **for** **(** i **=** 0 **;** i **<** 10 **;** i **++)** **{**  // ...  **}** |

#### Do not change the variable in the for loop stop condition inside the loop body or depend on the results returned from other code.

Variables in the for loop stop condition should not be changed inside the loop or depend on results returned from other code. This can lead to the risk of the loop never ending.

Non-compliant code example

|  |
| --- |
| **for** **(** int i **=** 0 **;** i **<** 10 **;** i **++)** **{**  **...**  i **=** i **-** 1 **;** // The counter variable is changed inside the loop  **...**  **}**  **for** **(** int i **=** 0 **;** i **<** getMaximumNumber **();** i **++)** **{...}** // The condition to stop the loop depends on the result returned from the function getMaximumNumber() |

Correct spelling:

|  |
| --- |
| int stopCondition **=** getMaximumNumber **();**  **for** **(** int i **=** 0 **;** i **<** stopCondition **;** i **++)** **{...}** |

#### Declare final with variable "public static"

There is no reason to declare a variable as "public" and "static" without "final". Thus, this value can be changed anywhere anywhere and can cause program errors.

Non-compliant code example

|  |
| --- |
| public class Greeter **{**  public static Foo foo **=** **newFoo** ( **);**  **...**  **}** |

Correct spelling:

|  |
| --- |
| public class Greeter **{**  public static final Foo foo **=** **newFoo** ( **);**  **...**  **}** |

#### Have the keyword "case" in each block of the switch declaration

Sometimes declaring a block of statements in a switch that does not start with the keyword case is still valid, but this makes the program difficult to understand and is often due to typing errors.

Non-compliant code example

|  |
| --- |
| **switch** **(** day **)** **{**  **case** MONDAY **:**  **case** TUESDAY **:**  WEDNESDAY **:** //syntactically correct but can lead to uncontrollably abnormal behavior  doSomething **();**  **break ;**  **...**  **}** |

Correct spelling:

|  |
| --- |
| **switch** **(** day **)** **{**  **case** MONDAY **:**  **break ;**  **case** TUESDAY **:**  foo **();**  **break ;**  **}** |

#### Do not pass a Collection object to its own method

This may cause program errors.

Non-compliant code example

|  |
| --- |
| List **<** Object **>** objs **=** **new** ArrayList **<** Object **>();**  objs **.** add **(** "Hello" **);**  objs **.** add **(** objs **);** // do not obey; throws a StackOverflowException exception if objs.hashCode() is called  objs **.** containsAll **(** objs **);** // do not obey; always return true  objs **.** removeAll **(** objs **);** // do not obey; easily confusing. should use clear() instead  **.....** |

#### Don't throw exception in Servlet

Throwing exceptions in the servlet can put the web server in unexpected states, potentially vulnerable to denial-of-service attacks.

Non-compliant code example

|  |
| --- |
| public void doGet **(** HttpServletRequest request **,** HttpServletResponse response **)**  **throws** IOException **,** ServletException **{**  String ip **=** request **.** getRemoteAddr **();**  InetAddress addr **=** InetAddress **.** getByName **(** ip **);** // Noncompliant; getByName(String) throws UnknownHostException  //...  **}** |

Correct spelling:

|  |
| --- |
| public void doGet **(** HttpServletRequest request **,** HttpServletResponse response **)**  **throws** IOException **,** ServletException **{**  **try** **{**  String ip **=** request **.** getRemoteAddr **();**  InetAddress addr **=** InetAddress **.** getByName **(** ip **);**  //...  **}**  **catch** **(** UnknownHostException uhex **)** **{**  //...  **}** |

#### Garbage cleanup is only done from the JVM

It is not recommended to call System.gc() or Runtime.getRuntime().gc() because it is impossible to know exactly what the JVM will do implicitly, it depends on the vendor, version and options:

* + Does the whole application freeze while calling these methods?
  + Is the -XX:DisableExplicitGC option enabled?
  + Will the JVM ignore calling these functions explicitly?
  + ...

The garbage collection task should be reserved for the JVM

#### Does not compare equals with data type Float

Mathematical operations on Float numbers are imprecise. Even when performing a series of operations on the Float, each run gives a different result, it depends on the Compiler and the Compiler settings.

Non-compliant code example

|  |
| --- |
| float zeroFloat **=** 0.0f **;**  **if** **(** zeroFloat **==** 0 **)** **{** // Irregular, return false  **}** |

Correct spelling:

|  |
| --- |
| float zeroFloat **=** 0.0f **;**  **if** **(** Float **.** floatToRawIntBits **(** zeroFloat **)** **==** 0 **)** **{** // True to the rules. Using bit comparison ensures we can compare with the value 0  **}** |

#### No access to static properties from Instance's methods

Updating a static field from a non-static method can easily lead to errors if there are multiple Class Instances or Threads running. Ideally, static fields should only be updated from static and synchronized methods.

Non-compliant code example

|  |
| --- |
| public class MyClass **{**  private static int count **=** 0 **;**  public void doSomething **()** **{**  //...  count **++;** // non-compliance, change the static property  **}**  **}** |

#### Initialize a static field set in "synchronized"

In a multi-threaded situation it may happen that the second process accesses an object that is being incompletely initialized by the first process. Allowing such access can cause fatal errors. The solution is that the initialization block for the static field should be synchronized or declare the variable as volatile.

Non-compliant code example

|  |
| --- |
| protected static Object instance **=** **null ;**  public static Object getInstance **()** **{**  **if** **(** instance **!=** **null )** **{**  **return** instance **;**  **}**  instance **=** **new** Object **();** // Not according to the rules  **return** instance **;**  **}** |

Correct spelling:

|  |
| --- |
| protected static volatile Object instance **=** **null ;**  public static Object getInstance **()** **{**  **if** **(** instance **!=** **null )** **{**  **return** instance **;**  **}**  instance **=** **new** Object **();**  **return** instance **;**  **}**  //or  protected static Object instance **=** **null ;**  public static synchronized Object getInstance **()** **{**  **if** **(** instance **!=** **null )** **{**  **return** instance **;**  **}**  instance **=** **new** Object **();**  **return** instance **;**  **}** |

#### Unleash the Lock

The logic in a method should ensure that the Lock is released in the methods that call it. Failure to release the Lock increases the risk of DeadLock and can cause runtime errors.

Non-compliant code example

|  |
| --- |
| public class MyClass **{**  Lock lock **=** **newLock** ( **);**  public void acquireLock **()** **{**  lock **.** lock **();** // not compliant, need to release right in this method.  **}**  public void releaseLock **()** **{**  lock **.** unlock **();**  **}**  public void doTheThing **()** **{**  acquireLock **();**  // do work...  releaseLock **();**  **}**  **}** |

Correct spelling:

|  |
| --- |
| public class MyClass **{**  Lock lock **=** **newLock** ( **);**  public void doTheThing **()** **{**  lock **.** lock **();**  // do work...  lock **.** unlock **();**  **}**  **}** |

#### Do not declare "public static" mutable fields

For mutable fields should not declare public static. That field must be passed into the class to reduce direct access. Avoid the risk of errors when multiple processes change the value at the same time.

Non-compliant code example

|  |
| --- |
| public interface MyInterface **{**  public static String **[]** strings **;** // do not obey  **}**  public class A **{**  public static String **[]** strings1 **=** **{** "first" **,** "second" **};** // do not obey  public static String **[]** strings2 **=** **{** "first" **,** "second" **};** // do not obey  public static List **<** String **>** strings3 **=** **new** ArrayList **<>();** // do not obey  // ...  **}** |

#### Do not use "Math.abs" for a possible value of "MIN\_VALUE"

There is a possibility that hashCode returns the value Integer.MIN\_VALUE, the absolute value of which can still be negative while the expected value is a positive number. This leads to unusual, unreliable results.

Non-compliant code example

|  |
| --- |
| public void doSomething **(** String str **)** **{**  **if** **(** Math **.** abs **(** str **.** hashCode **())** **>** 0 **)** **{** // non-compliant because str.hashCode() can be MIN\_VALUE  // ...  **}**  **}** |

Correct spelling:

|  |
| --- |
| public void doSomething **(** String str **)** **{**  **if** **(** str **.** hashCode **()** **!=** 0 **)** **{**  // ...  **}**  **}** |

#### Do not implement write non-serializable classes

Performing this operation may result in an exception.

Non-compliant code example

|  |
| --- |
| public class Vegetable **{** // do not implement Serializable or extend from the parent class that implements Serializable  //...  **}**  public class Menu **{**  public void meal **()** **throws** IOException **{**  Vegetable veg **;**  //...  FileOutputStream fout **=** **new** FileOutputStream **(** veg **.** getName **());**  ObjectOutputStream oos **=** **new** ObjectOutputStream **(** fout **);**  oos **.** writeObject **(** veg **);** // Not true, nothing is written to the file  **}**  **}** |

Correct spelling:

|  |
| --- |
| public class Vegetable **implements** Serializable **{** // this object can already be serialized  //...  **}**  public class Menu **{**  public void meal **()** **throws** IOException **{**  Vegetable veg **;**  //...  FileOutputStream fout **=** **new** FileOutputStream **(** veg **.** getName **());**  ObjectOutputStream oos **=** **new** ObjectOutputStream **(** fout **);**  oos **.** writeObject **(** veg **);**  **}**  **}** |

#### Eliminate test conditions that never occur in if/else.

The if/else if/else statement is checked from top to bottom, so at most only 1 branch is executed (when the test condition is true). Therefore, if the test conditions overlap or never occur, it will make the code difficult to understand, losing time to check when running the program.

Non-compliant code example

|  |
| --- |
| **if** **(** param **==** 1 **)** // First check  openWindow **();**  **else** **if** **(** param **==** 2 **)**  closeWindow **();**  **else** **if** **(** param **==** 1 **)** // Never run to here or always false  moveWindowToTheBackground **();**  **else** **if** **(** param **==** first **&&** param **==** 2 **)** // Never run to here or always value false  doSomethings **();**  **}** |

Correct spelling:

|  |
| --- |
| **if** **(** param **==** 1 **)**  openWindow **();**  **else** **if** **(** param **==** 2 **)**  closeWindow **();**  **else** **if** **(** param **==** 3 **)**  moveWindowToTheBackground **();**  **}** |

#### The return value of the function should not be ignored.

Test the code snippets that don't affect the entire program. If excess, proceed to remove.

Non-compliant code example

|  |
| --- |
| public void handle **(** String command **){**  command **.** toLowerCase **();** // Noncompliant; result of method thrown away  **...**  **}** |

Correct spelling:

|  |
| --- |
| public void handle **(** String command **){**  String formattedCommand **=** command **.** toLowerCase **();**  **...**  **}** |

#### Do not use non-short-circuit operators in conditional expressions.

Using non-short-circuit logic operations (||, &&) in the conditional expression can cause fatal errors in the program.

Non-compliant code example

|  |
| --- |
| **if (** getTrue **()** **|** getFalse **())** **{** **...** **}** // both conditions are evaluated |

Correct spelling:

|  |
| --- |
| **if (** getTrue **()** **||** getFalse **())** **{** **...** **}** |

#### End Switch cases with "break" command

End of a case if not break, the next case will continue to execute, so can cause uncontrollable error

Non-compliant code example

|  |
| --- |
| **switch** **(** myVariable **)** **{**  **case** 1 **:**  foo **();**  **break ;**  **case** 2 **:** // 'doSomething()' and 'doSomethingElse()' will both be executed, is this intentional or a typo?  doSomething **();**  **default :**  doSomethingElse **();**  **break ;**  **}** |

Correct spelling:

|  |
| --- |
| **switch** **(** myVariable **)** **{**  **case** 1 **:**  foo **();**  **break ;**  **case** 2 **:**  doSomething **();**  **break ;**  **default :**  doSomethingElse **();**  **break ;**  **}**  //This rule may not apply in the following cases:  **switch** **(** myVariable **)** **{**  **case** 0 **:** // empty case, used to represent a group of cases that share the same behavior  **case** 1 **:**  doSomething **();**  **break ;**  **case** 2 **:** // use return  **return ;**  **case** 3 **:** // use return  **throw** **new** IllegalStateException **();**  **default :** // for the last case, break is not required  doSomethingElse **();**  **}** |

#### Do not call Thread.run() and Runnable.run() directly

It doesn't make sense to call these methods directly because it will be executed on the current thread. Use the Thread.start() method instead

Non-compliant code example

|  |
| --- |
| Thread myThread **=** **new** Thread **(** runnable **);**  myThread **.** run **();** // Noncompliant |

Correct spelling:

|  |
| --- |
| Thread myThread **=** **new** Thread **(** runnable **);**  myThread **.** start **();** // Compliant |

#### Don't call Throwable.printStackTrace(...).

Throwable.printStackTrace(...) should not be used to print the log to the screen (only used in case of debugging). Instead, use more convenient logging libraries to take advantage of these advantages:

- Can easily retrieve logs

- Uniform log message format.

Non-compliant code example

|  |
| --- |
| **try** **{**  /\* ... \*/  **}** **catch (** Exception e **)** **{**  e **.** printStackTrace **();** // Not obeying the law  **}** |

Correct spelling:

|  |
| --- |
| **try** **{**  /\* ... \*/  **}** **catch (** Exception e **)** **{**  LOGGER **.** log **(** "context" **,** e **);** // obey the law  **}** |

#### When handling exception, it is necessary to preserve the original exception

When handling a caught exception, the messange and stack trace of the original exception should be logged and forwarded.

Non-compliant code example

|  |
| --- |
| **try** **{** /\* ... \*/ **}** **catch** **(** Exception e **)** **{** LOGGER **.** info **(** "context" **);** **}** //exceptions are not reserved    **try** **{** /\* ... \*/ **}** **catch** **(** Exception e **)** **{** LOGGER **.** info **(** e **.** getMessage **());** **}** //exception is not reserved (only can save message)  **try** **{** /\* ... \*/ **}** **catch** **(** Exception e **)** **{** **throw** **new** RuntimeException **(** "context" **);** **}** //exceptions are not reserved |

Correct spelling:

|  |
| --- |
| **try** **{** /\* ... \*/ **}** **catch** **(** Exception e **)** **{** LOGGER **.** info **(** e **);** **}**  **try** **{** /\* ... \*/ **}** **catch** **(** Exception e **)** **{** **throw** **new** RuntimeException **(** e **);** **}**  **try** **{**  /\* ... \*/  **}** **catch** **(** RuntimeException e **)** **{**  doSomething **();**  **throw** e **;**  **}** **catch** **(** Exception e **)** **{**  **throw** **new** RuntimeException **(** e **);** // allowed to convert exception  **}**  //Exception  InterruptedException **,** NumberFormatException **,** ParseException and MalformedURLException  int myInteger **;**  **try** **{**  myInteger **=** Integer **.** parseInt **(** myString **);**  **}** **catch** **(** NumberFormatException e **)** **{**  // it's perfectly acceptable to not deal with "e" here  myInteger **=** 0 **;**  **}** |

#### Conditional expressions should not always be "TRUE" or "FALSE"

The conditional expression always FALSE causes the next block of statements to never be called, similarly if always TRUE means the conditional clause is redundant and makes the code unreadable. These conditional expressions need to be removed or adjusted so that the condition is always TRUE or FALSE.

Non-compliant code example

|  |
| --- |
| //foo cannot be equal to and not equal to bar in the same conditional expression  **if (** foo **==** bar **&&** something **&&** foo **!=** bar **)** **{...}**  private void compute **(** int foo **)** **{**  **if** **(** foo **==** 4 **)** **{**  doSomething **();**  // foo equals 4, so this condition is always false  **if** **(** foo **>** 4 **)** **{...}**  **...**  **}**  **...**  **}**  private void compute **(** boolean foo **)** **{**  **if** **(** foo **)** **{**  **return ;**  **}**  doSomething **();**  // here foo is always false  **if** **(** foo **){...}**  **...**  **}** |

#### Do not declare static variables with objects of type "Calendars" and "DateFormats"

Calendar and DateFormat are not thread-safe. Using these objects when doing multithreading can lead to data problems or exceptions at runtime.

Non-compliant code example

|  |
| --- |
| public class MyClass **{**  static private SimpleDateFormat format **=** **new** SimpleDateFormat **(** "HH-mm-ss" **);** // Do not obey  static private Calendar calendar **=** Calendar **.** getInstance **();** // Do not obey |

Correct spelling:

|  |
| --- |
| public class MyClass **{**  private SimpleDateFormat format **=** **new** SimpleDateFormat **(** "HH-mm-ss" **);**  private Calendar calendar **=** Calendar **.** getInstance **();** |

#### Don't compare Class type by Class Name

There is no constraint that makes a class name unique (only in a package actually), so determining the type of an object based on a class name is dangerous and can lead to program errors. One of the risks is that an attacker could send objects with the same name as trusted classes and thereby gain exclusive access to the trusted class.

Use the instance of operator instead to check the object type.

Non-compliant code example

|  |
| --- |
| package computer **;**  class Pear **extends** Laptop **{** **...** **}**  package food **;**  class Pear **extends** Fruit **{** **...** **}**  class Store **{**  public boolean hasSellByDate **(** Object item **)** **{**  **if** **(** "Pear" **.** equals **(** item **.** getClass **().** getSimpleName **()))** **{** // do not obey  **return** **true ;**  **}**  **}**  **}** |

Correct spelling:

|  |
| --- |
| class Store **{**  public boolean hasSellByDate **(** Object item **)** **{**  **if** **(** item **instanceof** food **.** Pear **)** **{**  **return** **true ;**  **}**  **}**  **}** |

#### Do not use the same operator on both sides of a binary expression

This is usually due to a mistake (copy/paste) or simply redundant code

This argument does not apply to \*, +, and =

Non-compliant code example

|  |
| --- |
| **if** **(** a **==** a **)** **{** // always true  doZ **();**  **}**  **if** **(** a **!=** a **)** **{** // always false  doY **();**  **}**  **if** **(** a **==** b **&&** a **==** b **)** **{** // if the first is true, so is the second  doX **();**  **}**  **if** **(** a **==** b **||** a **==** b **)** **{** // if the first is true, so is the second  doW **();**  **}**  int j **=** 5 **/** 5 **;** //always 1  int k **=** 5 **-** 5 **;** //always 0 |

Correct spelling:

|  |
| --- |
| doZ **();**  **if** **(** a **==** b **)** **{**  doX **();**  **}**  **if** **(** a **==** b **)** **{**  doW **();**  **}**  int j **=** 1 **;**  int k **=** 0 **;**  //Exception  //Compare a float with itself to check for NaN  //Similarly shifting 1 bit to 1 is a common way to create bit masks.  float f **;**  **if (** f **!=** f **)** **{** //check the value of NaN  System **.** out **.** println **(** "f is NaN" **);**  **}**  int i **=** first **<<** 1 **;** // Valid  int j **=** a **<<** a **;** // illegal |

#### Removed "dead stores"

Dead store is one of the following two cases:

* + Declare a local variable and assign a value to it (including null), but then not use it anywhere.
  + Calculates and retrieves a value but then does not use it.

These two cases can often lead to fatal errors, even without errors consuming unnecessary resources.

Non-compliant code example

|  |
| --- |
| public void pow **(** int a **,** int b **)** **{**  **if (** b **==** 0 **)** **{**  **return** 0 **;**  **}**  int x **=** a **;**  **for (** int i **=** 1 **,** i **<** b **,** i **++)** **{**  x **=** x **\*** a **;** //this value is not used anywhere  **}**  **return** a **;**  **}** |

Correct spelling:

|  |
| --- |
| public void pow **(** int a **,** int b **)** **{**  **if (** b **==** 0 **)** **{**  **return** 0 **;**  **}**  int x **=** a **;**  **for (** int i **=** 1 **,** i **<** b **,** i **++)** **{**  x **=** x **\*** a **;**  **}**  **return** x **;**  **}** |

#### When testing for equality with a String variable, Strings should be placed to the left of the equal . comparison expression

It is recommended to place the String string on the left side of the equals() or equalsIgnoreCase() method, this can prevent null pointer exceptions from occurring because a string will always be non-null.

Non-compliant code example

|  |
| --- |
| String myString **=** **null ;**  System **.** out **.** println **(** "Equal?" **+** myString **.** equals **(** "foo" **));** // Error null pointer exception  System **.** out **.** println **(** "Equal?" **+** **(** myString **!=** **null** **&&** myString **.** equals **(** "foo" **)));** // cumbersome and complicated way of writing |

Correct spelling:

|  |
| --- |
| System **.** out **.** println **(** "Equal?" **+** "foo" **.** equals **(** myString **));** // for short, solves the null case |

#### Use ConcurrentHashMap instead of HashMap

From Java5, ConcurrentHashMap is installed, specially designed for parallel, multi-threaded applications. Should be used as an alternative to HashMap or classes that implement interace Map.

#### Use pool when calling webservice

When java application calls webservice of another system need to use pool

Example using PoolingHttpClientConnectionManager in HttpClient library

Create class HttpClientFactory to manage

|  |
| --- |
| **import** org.apache.http.client.config.RequestConfig;  **import** org.apache.http.impl.client.CloseableHttpClient;  **import** org.apache.http.impl.client.HttpClients;  **import** org.apache.http.impl.conn.PoolingHttpClientConnectionManager;   **import** javax.net.ssl.SSLContext;  **import** java.util.logging.Level;  **import** java.util.logging.Logger;   */\*\*  \* \** ***@author***  *\** ***@created*** *Feb 11, 2017  \* \*/* **public class** HttpClientFactory { **private static volatile** PoolingHttpClientConnectionManager *cm* ;    **private static void** initPool() **throws** Exception{  **if** ( *cm* == **null** ){  *cm* = **new** PoolingHttpClientConnectionManager();  *cm* .setMaxTotal( 300 );  *cm* .setDefaultMaxPerRoute( 100 );  } } **static** {  **try** {  *initPool* ();  } **catch** (Exception ex) {  Logger. *getLogger* (HttpClientFactory. **class .getName** ()).log(Level. ***SEVERE*** , **null** , ex);  } } **public static** CloseableHttpClient getHttpClientFromPool() **throws** Exception {  RequestConfig requestConfig = RequestConfig. *custom* ()  .setConnectTimeout( 30000 )  .setSocketTimeout( 30000 ).build( );  CloseableHttpClient client = HttpClients. *custom* ().setConnectionManager( *cm* )  .setDefaultRequestConfig(requestConfig) .setConnectionManagerShared( **true** )  .disableAutomaticRetries() .build();   **return** client;  }   } |

Use the CloseableHttpClient object to call the webservice:

|  |
| --- |
| **public static** String sendMessageToAdapter(BusinessAdapterDTO baseMessage) **throws** Exception {   String jsonString = MappingObjectJson. *convertObjectToJson* (baseMessage);  String username = **""** ;  **if** (baseMessage!= **null** &&baseMessage.getSender\_data()!= **null** ){  username = baseMessage.getSender\_data().getUsername(); } **long** startTime = System. *currentTimeMillis* ();  *logger* .info( **"username: "** +username+ **", Input BusinessAdapter: "** +jsonString);  String url = *getBusinessAdapterUrl* () + **"/api/business\_adapter/runService"** ;  String result = **null** ;  HttpPost post = **new** HttpPost(url);  StringEntity params = **new** StringEntity(jsonString, ContentType. ***APPLICATION\_JSON*** );  post.addHeader( **"content-type"** , **"application/json; charset=utf-8"** );  post.setEntity(params); post.setHeader( **"USERNAME"** , *getUserWS* ());  post.setHeader( **"PASSWORD"** , *getPassWS* ());  **try** (CloseableHttpClient httpclient = HttpClientFactory. *getHttpClientFromPool* (); CloseableHttpResponse HttpResponse = httpclient.execute(post)) {  result = EntityUtils. *toString* (httpResponse.getEntity());  **long** endTime = System. *currentTimeMillis* ();  *logger* .info( **"username: "** +username+ **". finished call BusinessAdapter:"** + result+ **", process time: "** + String. *valueOf* (endTime - startTime) + **" milisecond"** );  } **return** result;  } |

#### Ensure the performance of updating data to redis using jedis . library

When updating data to redis avoid calling sync command (for example using jedis.sync in jedis library) especially in case of system writing large data objects when using this command will lead to node master of redis immediately synchronizes the data to the slave nodes, which slows down the redis cluster.

Wrong code example (correct need to remove the line jedis.sync() )

|  |
| --- |
| **public static void** saveUserToServer(UserData userData, String server) {  Jedis jedis = **null** ;  **try** {  ***logger*** .info( **"----> Start saveUserToServer"** );  jedis = JedisSentinelConnectionPool. *getRedis* ();  jedis.select(Constants.REDIS. ***DB\_DEFAULT*** );  String now = Util. *getSysdate* ( **"yyyyMMddHHmmss"** );  jedis.hset(Constants.REDIS.SERVER\_GROUP + ***userData.getUsername*** (), server, now);  ***logger*** .info( **"----> End saveUserToServer suscess"** );  jedis.sync();  } **catch** (Exception e) {  ***logger*** .error( **"----> saveUserToServer have error: "** , e);  } **finally** {  JedisSentinelConnectionPool. *closeJedis* (jedis);  }} |

#### Note when using synchronized

This section adds some experience about using synchronized beyond the content in item 2. Do not use "Lock" in synchronized block

It gives examples to limit the use of synchronized

+ Functions calling webservices of other systems are not synchronized

Example of non-conforming code (correct need to remove **synchronized** )

|  |
| --- |
| **public static synchronized** BaseChatbotResponse sendMessageToBot(String wsAddress, JSONObject baseMessage) {  String jsonString = baseMessage.toString(); String url = wsAddress + **"/receiveQuestion"** ;  BaseChatbotResponse result = **null** ;  Map<String, String> headers = **new** HashMap<>();  **try** {String result2 = *sendMessageToRestapi* (url,jsonString,headers);  **if** (Constants.MESSAGE. ***TIMEOUT\_MESSAGE*** .equalsIgnoreCase(result2)) {  String token = MemoryDataLoaderRedis. *getTokenByUsername* (baseMessage.getJSONObject( **"userData"** ).getString( **"username"** ));  OkHttpCaller. *sendMessageToEContact* (token, baseMessage.getJSONObject( **"userData"** ).getString( **"staffCode"** ), **null** , **false** );  **return** result;  } *logger* .info( **"Ket via ws:"** + result2);  result = *gson* .fromJson(result2, BaseChatbotResponse. **class** );  *logger* .info( **"Pass after parser:"** + result);  **return** result;  } **catch** (Exception ex) {  *logger* .error(ex.getMessage(), ex);  } **return** result;   } |

+ When the code has to update the value into a variable / object, consider using objects that support multi thread concurrency.

* For example, in case the system needs to save a file to manage the number of concurrent transactions in a webservice, it is necessary to write a WebserviceUtils class that has a variable **totalCurrentRequest** to count connections. When starting the transaction, it will run into the function increase and increase variable **totalCurrentRequest** 1 unit. At the end of the transaction will run into the functionreleaseRequest to decrement the **totalCurrentRequest variable by** 1, so **totalCurrentRequest** is the number of concurrent transactions.
* If using synchronized

|  |
| --- |
| **import** java.util.concurrent.atomic.AtomicInteger;  **import** org.apache.log4j.Logger;   **public class** WebserviceUtils {   **private** Logger **logger** ;  **private** Integer **totalCurrentRequest** = **0** ;  **private** Long **maxConnection** ;**private** String **wsCode** ;   **public** WebserviceUtils(Long maxConnection, Logger logger) {  **this** . **maxConnection** = maxConnection;  **this** . **logger** = logger;  }   */\*\*  \* Tang request len 1 \*/* **public** *synchronized* **void** increase() {  **totalCurrentRequest** ++;  }   */\*\*  \* Request stage \*/* **public** *synchronized* **void** releaseRequest() {  **if** ( **totalCurrentRequest** > 0 ) {  **totalCurrentRequest** --;  } }   } |

Code using AtomicInteger for faster processing speed when there are many transactions:

|  |
| --- |
| **import** java.util.concurrent.atomic.AtomicInteger;  **import** org.apache.log4j.Logger;   **public class** WebserviceUtils {   **private** Logger **logger** ;  **private** AtomicInteger **totalCurrentRequest** = **new** AtomicInteger( 0 );  **private** Long **maxConnection** ;**private** String **wsCode** ;   **public** WebserviceUtils(Long maxConnection, Logger logger) {  **this** . **maxConnection** = maxConnection;  **this** . **logger** = logger;  }   */\*\*  \* Tang request len 1 \*/* **public void** increase() {  **totalCurrentRequest** .incrementAndGet();  }   */\*\*  \* Request stage \*/* **public void** releaseRequest() {  **if** ( **totalCurrentRequest** .get() > 0 ) {  **totalCurrentRequest** .decrementAndGet();  } }} |

# APPENDIX 02: PROGRAMMING GUIDELINES AVOID POTENTIAL ERRORS .NET APPLICATION CODE

#### Do not return the "IDisposables" object initialized in the code using "using".

The object "IDisposables" after going out of scope of the using code will be automatically released. Therefore, if you want to return the variable value that has not been released, you should remove the "using" statement or do not return the "IDisposables" object.

Non-compliant code example

|  |
| --- |
| **public** FileStream WriteToFile **(** string path **,** stringtext **)** \_  **{**  **using** **(** var fs **=** File .Create **(** path **) )** // Noncompliant  **{**  var bytes **=** Encoding **.** UTF8 **.** GetBytes **(** text **);**  fs **.** Write **(** bytes **,** 0 **,** bytes **.** Length **);**  **return** fs **;**  **}**  **}** |

Correct spelling:

|  |
| --- |
| **public** FileStream WriteToFile **(** string path **,** stringtext **)** \_  **{**  var fs **=** File **.** Create **(** path **);**  var bytes **=** Encoding **.** UTF8 **.** GetBytes **(** text **);**  fs **.** Write **(** bytes **,** 0 **,** bytes **.** Length **);**  **return** fs **;**  **}** |

#### Use short-circuit logic to avoid null memory access errors in comparison conditions

When one condition checks for null, further testing the remaining conditions may result in a null pointer error

Non-compliant code example

|  |
| --- |
| **if** **(** str **==** **null** **&&** str **.** Length **==** 0 **)**  **{**  Console **.** WriteLine **(** "String is empty" **);**  **}**  **if** **(** str **!=** **null** **||** str **.** Length **>** 0 **)**  **{**  Console **.** WriteLine **(** "String is not empty" **);**  **}** |

Correct spelling:

|  |
| --- |
| **if** **(** str **==** **null** **||** str **.** Length **==** 0 **)**  **{**  Console **.** WriteLine **(** "String is empty" **);**  **}**  **if** **(** str **!=** **null** **&&** str **.** Length **>** 0 **)**  **{**  Console **.** WriteLine **(** "String is not empty" **);**  **}** |

#### Objects that implement the "IDisposables" interface need to be disposed.

In many cases it is not possible to rely solely on Garbage collection to clean things up, for example not being able to release non-memory resources like Files. For this case, IDisposable should be used and the Dispose method will always be called to release the object.

When a class variable is an "IDisposable" object, the object should be called dispose in the Class's Dispose method to ensure that the object is automatically released when the class is released. If IDisposable is a local variable, it should be initialized using “using”.

Non-compliant code example

|  |
| --- |
| **public** class ResourceHolder  **{**  **private** FileStream fs **;** // fs is never released  **public** void OpenResource **(** string path **)**  **{**  **this .** fs **=** **new** FileStream **(** path **,** FileMode **.** Open **);**  **}**  **public** void CloseResources **()**  **{**  **this .** fs **.** Close **();**  **}**  **public** void WriteToFile **(** string path **,** stringtext **)** \_  **{**  var fs **=** **new** FileStream **(** path **,** FileMode **.** Open **);** // Noncompliant  var bytes **=** Encoding **.** UTF8 **.** GetBytes **(** text **);**  fs **.** Write **(** bytes **,** 0 **,** bytes **.** Length **);**  **}**  **}** |

Correct spelling:

|  |
| --- |
| **public** class ResourceHolder **:** IDisposable  **{**  **private** FileStream fs **;**  **public** void OpenResource **(** string path **)**  **{**  **this .** fs **=** **new** FileStream **(** path **,** FileMode **.** Open **);**  **}**  **public** void CloseResources **()**  **{**  **this .** fs **.** Close **();**  **}**  **public** void Dispose **()**  **{**  **this .** fs **.** Dispose **();**  **}**  **public** void WriteToFile **(** string path **,** stringtext **)** \_  **{**  **using** **(** var fs **=** **new** FileStream **(** path **,** FileMode **.** Open **))**  **{**  var bytes **=** Encoding **.** UTF8 **.** GetBytes **(** text **);**  fs **.** Write **(** bytes **,** 0 **,** bytes **.** Length **);**  **}**  **}**  **}**  //Attention:  // The local variable "IDisposables" returned in the return statement is not // freed, it is the same as a non-local variable.  **public** Stream WriteToFile **(** string path **,** stringtext **)** \_  **{**  var fs **=** **new** FileStream **(** path **,** FileMode **.** Open **);**  var bytes **=** Encoding **.** UTF8 **.** GetBytes **(** text **);**  fs **.** Write **(** bytes **,** 0 **,** bytes **.** Length **);**  **return** fs **;**  **}** |

#### The increment variable in the "for" loop must be the one in the loop stop condition

When the increment variable and the for loop stop condition variable are not the same, it is often an error that can lead to the loop never ending and if not causing an error, it is very difficult to maintain later.

Non-compliant code example

|  |
| --- |
| **for** **(** i **=** 0 **;** i **<** 10 **;** j **++)**  **{**  // ...  **}** |

Correct spelling:

|  |
| --- |
| **for** **(** i **=** 0 **;** i **<** 10 **;** i **++)**  **{**  // ...  **}** |

#### Anonymous Delegate cannot be used to unsubscribe from an event.

Events can be subscribed to anonymous delegates (anonymous i.e. not tied to variables), but in doing so it is not possible to unsubscribe because: during registration/cancellation trust subscriptions will be added/removed from a list, both processes are anonymous so that upon unsubscribing it will not be possible to identify the previously registered trust.

If you want to unregister the event label, you should declare the trust as a specific variable and use that variable.

Non-compliant code example

|  |
| --- |
| listView **.** PreviewTextInput **+=** **(** obj **,** args **)** **=>**  listView\_PreviewTextInput **(** obj **,** args **,** listView **);** **(** event label **)**  // ...  listView **.** PreviewTextInput **-=** **(** obj **,** args **)** **=>**  listView\_PreviewTextInput **(** obj **,** args **,** listView **);** // (unlabeling) |

Correct spelling:

|  |
| --- |
| EventHandler func **=** **(** obj **,** args **)** **=>** listView\_PreviewTextInput **(** obj **,** args **,** listView **);** **(** label the event as a variable func **)**  listView **.** PreviewTextInput **+=** func **;** **(** event label **)**  // ...  listView **.** PreviewTextInput **-=** func **;** **(** unlabeling **)** |

#### The class must release member variables in that Class's Dispose function.

In order for member variables to be automatically released when a Class is released, it is recommended to release member variables in the Dispose function of that Class.

Non-compliant code example

|  |
| --- |
| **public** class ResourceHolder **:** IDisposable  **{**  **private** FileStream fs **;**  **public** void OpenResource **(** string path **)**  **{**  **this .** fs **=** **new** FileStream **(** path **,** FileMode **.** Open **);**  **}**  **public** void CloseResources **()**  **{**  **this .** fs **.** Close **();**  **}**  **public** void CleanUp **()**  **{**  **this .** fs **.** Dispose **();** // Noncompliant; Dispose not called in class' Dispose method  **}**  **public** void Dispose **()**  **{**  // method added to satisfy demands of interface  **}**  **}** |

Correct spelling:

|  |
| --- |
| **public** class ResourceHolder **:** IDisposable  **{**  **private** FileStream fs **;**  **public** void OpenResource **(** string path **)**  **{**  **this .** fs **=** **new** FileStream **(** path **,** FileMode **.** Open **);**  **}**  **public** void CloseResources **()**  **{**  **this .** fs **.** Close **();**  **}**  **public** void Dispose **()**  **{**  **this .** fs **.** Dispose **();**  **}**  **}** |

#### If a class has a member variable of "IDisposable", that class should implement the "IDisposable" interface.

If the class has a member variable of "IDisposable", so that the member variables are automatically released when the class is released, the class should implement the "IDisposable" interface.

Non-compliant code example

|  |
| --- |
| **public** class ResourceHolder // Noncompliant; doesn't implement IDisposable  **{**  **private** FileStream fs **;** // This member is never Dispose'd  **public** void OpenResource **(** string path **)**  **{**  **this .** fs **=** **new** FileStream **(** path **,** FileMode **.** Open **);**  **}**  **public** void CloseResources **()**  **{**  **this .** fs **.** Close **();**  **}**  **}** |

Correct spelling:

|  |
| --- |
| **public** class ResourceHolder **:** IDisposable  **{**  **private** FileStream fs **;**  **public** void OpenResource **(** string path **)**  **{**  **this .** fs **=** **new** FileStream **(** path **,** FileMode **.** Open **);**  **}**  **public** void CloseResources **()**  **{**  **this .** fs **.** Close **();**  **}**  **public** void Dispose **()**  **{**  **this .** fs **.** Dispose **();**  **}**  **}** |

#### Do not log to Console

Logging Debug information is extremely important in development, but logging to Console in release (production), especially code running on the client side (client side) will risk exposing sensitive information. cold.

Non-compliant code example

|  |
| --- |
| **private** voidDoSomething ( **)**  **{**  // ...  Console **.** WriteLine **(** "so far, so good..." **);** // Not obeying the law  // ...  **}** |

#### Don't virtualize Field-like events

Field-like events are events that do not have explicit add and remove methods, the compiler will automatically generate a private delegate field, as well as add and remove methods.

When the event field-like is overridden by another field-like event, it generates a delegate field in the derived class, which exists in parallel with the fields in the parent class. The result is that too many separate events are generated (usually not by the programmer's intention). To avoid this error, the events filed-like should not be virtual

Non-compliant code example

|  |
| --- |
| **abstract** class Car  **{**  **public** **virtual** **event** EventHandler OnRefueled **;** // Noncompliant  **public** void Refuel **()**  **{**  // This OnRefueled will always be null  **if** **(** OnRefueled **!=** **null )**  **{**  OnRefueled **( this ,** **null );**  **}**  **}**  **}**  class R2 **:** Car  **{**  **public** **override** **event** EventHandler OnRefueled **;**  **}**  class Program  **{**  static void Main **(** string **[]** args **)**  **{**  var r2 **=** **new** R2 **();**  r2 **.** OnRefueled **+=** **new** EventHandler **((** o **,** a **)** **=>**  **{**  Console **.** WriteLine **(** "This event will never be called" **);**  **});**  r2 **.** refuel **();**  **}**  **}** |

Correct spelling:

|  |
| --- |
| **abstract** class Car  **{**  **public** **event** EventHandler OnRefueled **;** // Compliant  **public** void Refuel **()**  **{**  **if** **(** OnRefueled **!=** **null )**  **{**  OnRefueled **( this ,** **null );**  **}**  **}**  **}**  class R2 **:** Car **{}**  class Program  **{**  static void Main **(** string **[]** args **)**  **{**  var r2 **=** **new** R2 **();**  r2 **.** OnRefueled **+=** **new** EventHandler **((** o **,** a **)** **=>**  **{**  Console **.** WriteLine **(** "This event will be called" **);**  **});**  r2 **.** refuel **();**  **}**  **}** |

#### K does not compare unassigned generic type parameter with null

When a Generic parameter has not been assigned a specific type, it should not be compared with null, because the comparison result is always false because the parameter can be null but never null. Otherwise the Generic parameter needs to be assigned a specific type.

Non-compliant code example

|  |
| --- |
| **private** bool IsDefault **<** T **>(** T value **)**  **{**  **if** **(** value **==** **null )** // Noncompliant  **{**  // ...  **}**  // ...  **}** |

Correct spelling:

|  |
| --- |
| **private** bool IsDefault **<** T **>(** T value **)**  **{**  **if ( object .** Equals **(** value **,** **default (** T **)))**  **{**  // ...  **}**  // ...  **}**  Or  **private** bool IsDefault **<** T **>(** T value **)** where T **:** class  **{**  **if** **(** value **==** **null )**  **{**  // ...  **}**  // ...  **}** |

#### Expressions that are always true should not be used

Expressions that are always true should not be used

Non-compliant code example

|  |
| --- |
| **if** **(** a **==** a **)** // always true  **{**  doZ **();**  **}**  **if** **(** a **!=** a **)** // always false  **{**  doY **();**  **}**  **if** **(** a **==** b **&&** a **==** b **)** // if the first one is true, the second one is too  **{**  doX **();**  **}**  **if** **(** a **==** b **||** a **==** b **)** // if the first one is true, the second one is too  **{**  doW **();**  **}**  int j **=** 5 **/** 5 **;** //always 1  int k **=** 5 **-** 5 **;** //always 0 |

Correct spelling:

|  |
| --- |
| doZ **();**  **if** **(** a **==** b **)**  **{**  doX **();**  **}**  **if** **(** a **==** b **)**  **{**  doW **();**  **}**  int j **=** 1 **;**  int k **=** 0 **;**  //Exception  Bit shifting 1 is the usual way to generate bit masks, so the rule does not apply  int i **=** first **<<** 1 **;** // Compliant  int j **=** a **<<** a **;** // Noncompliant |

#### It is not recommended to use "OrderBy" consecutively in LinQ

It is not recommended to use OrderBy consecutively in LinQ because each OrderBy will reorder the entire list regardless of the result of the previous OrderBy.

Non-compliant code example

|  |
| --- |
| var x **=** personList  **.** OrderBy **(** person **=>** person **.** Age **)**  **.** OrderBy **(** person **=>** person **.** Name **)** // Noncompliant  **.** ToList **();** // x is sorted by Name, not sub-sorted |

Correct spelling:

|  |
| --- |
| var x **=** personList  **.** OrderBy **(** person **=>** person **.** Age **)**  **.** ThenBy **(** person **=>** person **.** Name **)**  **.** ToList **();** |

#### The "if/else if" structure should not have the same conditions

A sequence of "if/else if" constructs are evaluated from the top, usually only the first branch with true condition is executed. Therefore, repeating a condition (often by mistake when copying/pasting) can cause serious errors that we do not anticipate.

Non-compliant code example

|  |
| --- |
| **if** **(** param **==** 1 **)**  openWindow **();**  **else** **if** **(** param **==** 2 **)**  closeWindow **();**  **else** **if** **(** param **==** 1 **)** // error not following the rules  moveWindowToTheBackground **();** |

Correct spelling:

|  |
| --- |
| **if** **(** param **==** 1 **)**  openWindow **();**  **else** **if** **(** param **==** 2 **)**  closeWindow **();**  **else** **if** **(** param **==** 3 **)**  moveWindowToTheBackground **();** |

#### Do not use Static fields in objects of generic type

A static variable in a generic object is not shared in closed initialization derivatives e.g. the static variable in LengthLimitedSingletonCollection<int> and LengthLimitedSingletonCollection<string> points to two different objects

If you need to share static variables between instances (instances) with different types of generic parameters, you need to define a non-generic base class to store static members, then set the generic type to inherit from the class. basis.

Non-compliant code example

|  |
| --- |
| **public** class LengthLimitedSingletonCollection **<** T **>** where T **:** **new ()**  **{**  **protected** const int MaxAllowedLength **=** 5 **;**  **protected** static Dictionary **<** Type **,** **object >** instances **=** **new** Dictionary **<** Type **,** **object >();** // Noncompliant  **public** static T GetInstance **()**  **{**  **object** instance **;**  **if** **(!** instances **.** TryGetValue **( typeof (** T **),** **out** instance **))**  **{**  **if** **(** instances **.** Count **>=** MaxAllowedLength **)**  **{**  **throw** **newException** ( **);**  **}**  instance **=** **new** T **();**  instances **.** Add **( typeof (** T **),** instance **);**  **}**  **return** **(** T **)** instance **;**  **}**  **}** |

Correct spelling:

|  |
| --- |
| **public** class SingletonCollectionBase  **{**  **protected** static Dictionary **<** Type **,** **object >** instances **=** **new** Dictionary **<** Type **,** **object >();**  **}**  **public** class LengthLimitedSingletonCollection **<** T **>** **:** SingletonCollectionBase where T **:** **new ()**  **{**  **protected** const int MaxAllowedLength **=** 5 **;**  **public** static T GetInstance **()**  **{**  **object** instance **;**  **if** **(!** instances **.** TryGetValue **( typeof (** T **),** **out** instance **))**  **{**  **if** **(** instances **.** Count **>=** MaxAllowedLength **)**  **{**  **throw** **newException** ( **);**  **}**  instance **=** **new** T **();**  instances **.** Add **( typeof (** T **),** instance **);**  **}**  **return** **(** T **)** instance **;**  **}**  **}**  //Exception  If the static field or property uses a type parameter, then the developer is assumed to understand that the static member is not shared among the closed constructed types.  **public** class Cache **<** T **>**  **{**  **private** static Dictionary **<** string **,** T **>** CacheDictionary **{** get **;** set **;** **}** // Compliant  **}** |

# APPENDIX 03: PROGRAMMING GUIDELINES TO AVOID POTENTIAL ERROR CODES JAVASCRIPT APPLICATION CODE, HTML

#### Do not use "NaN" in comparison operations

The NaN value is not equal to any other value even itself. The isNaN() function should instead be used to compare the value of a variable with a numeric type.

|  |
| --- |
| **-** isNaN **(** a **)** **=** ***false*** **=>** a is numeric type  **-** isNaN **(** a **)** **=** ***true*** **=>** a is not a numeric type  isNaN **(** 123 **)** //false  isNaN **(-** 1.23 **)** //false  isNaN **(** 5 **-** 2 **)** //false  isNaN **(** 0 **)** //false  isNaN **(** '123' **)** //false  isNaN **(** 'Hello' **)** //true  isNaN **(** '2005/12/12' **)** //true  isNaN **(** '' **)** //false  isNaN **( *true* )** //false  isNaN **(** undefined **)** //true  isNaN **(** 'NaN' **)** //true  isNaN **(** NaN **)** //true  isNaN **(** 0 **/** 0 **)** //true |

Non-compliant code example:

|  |
| --- |
| ***var*** a **=** NaN **;**  ***if*** **(** a **===** NaN **)** **{** // always return false  console **.** log **(** "a is not a number" **);**  **}**  ***if*** **(** a **!==** NaN **)** **{** // always return true  console **.** log **(** "a is not NaN" **);**  **}** |

Correct spelling:

|  |
| --- |
| ***if*** **(** isNaN **(** a **)** **)** **{**  console **.** log **(** "a is not a number" **);**  **}** ***else*** **{**  console **.** log **(** "a is not NaN" **);**  **}** |

#### Use short-circuit logic to avoid null access errors in comparison conditions.

When one condition checks for null, further testing of the remaining conditions may result in a TypeError.

Non-compliant code example

|  |
| --- |
| ***if*** **(** str **==** null **&&** str **.** length **==** 0 **)** **{**  console **.** log **(** "String is empty" **);**  **}**  ***if*** **(** str **==** undefined **&&** str **.** length **==** 0 **)** **{**  console **.** log **(** "String is empty" **);**  **}**  ***if*** **(** str **!=** null **||** str **.** length **>** 0 **)** **{**  console **.** log **(** "String is not empty" **);**  **}**  ***if*** **(** str **!=** undefined **||** str **.** length **>** 0 **)** **{**  console **.** log **(** "String is not empty" **);**  **}** |

Correct spelling:

|  |
| --- |
| ***if*** **(** str **!=** null **&&** str **.** length **==** 0 **)** **{**  console **.** log **(** "String is empty" **);**  **}**  ***if*** **(** str **!=** undefined **&&** str **.** length **==** 0 **)** **{**  console **.** log **(** "String is empty" **);**  **}**  ***if*** **(** str **==** null **||** str **.** length **>** 0 **)** **{**  console **.** log **(** "String is not empty" **);**  **}**  ***if*** **(** str **==** undefined **||** str **.** length **>** 0 **)** **{**  console **.** log **(** "String is not empty" **);**  **}** |

#### Do not use the "delete" operator with type Array

The delete operator is used to remove a property from an object. In case the object is of type Array, the delete operator is also used similarly, but if this operator is used, the index of the elements after the deleted element does not jump to the index of the previous position.

To delete an element in the array and the following elements are pushed to the front to fill the position of the deleted element, the following functions should be used:

Array.prototype.splice - add/remove elements in array

Array.prototype.pop - add/remove the last element of the array

Array.prototype.shift - add/remove elements at the beginning of the array

Non-compliant code example

|  |
| --- |
| ***var*** myArray **=** **[** 'a' **,** 'b' **,** 'c' **,** 'd' **];**  ***delete*** myArray **[** 2 **];** // Array result: myArray => ['a', 'b', undefined, 'd']  console **.** log **(** myArray **[** 2 **]);** // Print output: undefined |

Correct spelling:

|  |
| --- |
| ***var*** myArray **=** **[** 'a' **,** 'b' **,** 'c' **,** 'd' **];**  // Remove element at index position = 2  removed **=** myArray **.** splice **(** 2 **,** 1 **);** // Array result: myArray => ['a', 'b', 'd']  console **.** log **(** myArray **[** 2 **]);** // Print result: 'd' |

#### Do not manipulate data with "eval" and "arguments" variables.

In Javascript, the eval() function is used to calculate values. Arguments are used to access parameters via index. Manipulating data with these objects can lead to unexpected errors.

Non-compliant code example

|  |
| --- |
| eval **=** 17 **;** // Noncompliant  arguments **++;** // Noncompliant  **++** eval **;** // Noncompliant  ***var*** obj **=** **{** set p **(** arguments **)** **{** **}** **};** // Noncompliant  ***var*** eval **;** // Noncompliant  ***try*** **{** **}** ***catch*** **(** arguments **)** **{** **}** // Noncompliant  ***function*** x **(** eval **)** **{** **}** // Noncompliant  ***function*** arguments **()** **{** **}** // Noncompliant  ***var*** y **=** ***function*** eval **()** **{** **};** // Noncompliant  ***var*** f **=** ***new*** Function **(** "arguments" **,** "return 17;" **);** // Noncompliant  ***function*** fun **()** **{**  ***if*** **(** arguments **.** length **==** 0 **)** **{** // Compliant  // do something  **}**  **}** |

Correct spelling:

|  |
| --- |
| result **=** 17 **;**  args **++;**  **++** result **;**  ***var*** obj **=** **{** set p **(** arg **)** **{** **}** **};**  ***var*** result **;**  ***try*** **{** **}** ***catch*** **(** args **)** **{** **}**  ***function*** x **(** arg **)** **{** **}**  ***function*** args **()** **{** **}**  ***var*** y **=** ***function*** fun **()** **{** **};**  ***var*** f **=** ***new*** Function **(** "args" **,** "return 17;" **);**  ***function*** fun **()** **{**  ***if*** **(** arguments **.** length **==** 0 **)** **{**  // do something  **}**  **}** |

#### The increment variable in the "for" loop must be the one in the loop stop condition

When the increment variable and the for loop stop condition variable are not the same, it is often an error that can lead to the loop never ending and if not causing an error, it is very difficult to maintain later.

Non-compliant code example

|  |
| --- |
| ***for*** **(** i **=** 0 **;** i **<** 10 **;** j **++)** **{** // Noncompliant  // ...  **}** |

Correct spelling:

|  |
| --- |
| ***for*** **(** i **=** 0 **;** i **<** 10 **;** i **++)** **{**  // ...  **}** |

#### Control when sending messages via text

HTML5 allows sending messages from an HTML page to an HTML page in a different domain. To avoid the risk of exposing sensitive information when sending to an unsecured domain, the data sent in the postMessage() function needs to be censored before sending.

Non-compliant code example

|  |
| --- |
| ***var*** myWindow **=** document **.** getElementById **(** 'myIFrame' **).** contentWindow **;**  myWindow **.** postMessage **(** message **,** "\*" **);** // Data in 'myIFrame' before sending may contain sensitive information? |

#### Remove the "debugger" command before deploying

"debugger" is a command used by programmers to find bugs during application development. After product packaging in the deployment phase all 'debugger' statements need to be removed from the source code.

Non-compliant code example

|  |
| --- |
| ***for*** **(** i **=** 1 **;** i **<** 5 **;** i **++)** **{**  Debug **.** write **(** "loop index is " **+** i **);**  ***debugger* ;**  **}** |

Correct spelling:

|  |
| --- |
| ***for*** **(** i **=** 1 **;** i **<** 5 **;** i **++)** **{**  Debug **.** write **(** "loop index is " **+** i **);**  **}** |

#### Don't use obvious expressions that are always true or always false

This is often the case by mistake (copy/paste) or simply redundant code, making it difficult to maintain.

Non-compliant code example

|  |
| --- |
| ***if*** **(** a **==** a **)** **{** // always true  doZ **();**  **}**  ***if*** **(** a **!=** a **)** **{** // always wrong  doY **();**  **}**  ***if*** **(** a **==** b **&&** a **==** b **)** **{**  doX **();**  **}**  ***if*** **(** a **==** b **||** a **==** b **)** **{**  doW **();**  **}**  ***var*** j **=** 5 **/** 5 **;** //always = 1  ***var*** k **=** 5 **-** 5 **;** //always = 0 |

Correct spelling:

|  |
| --- |
| doZ **();**  ***if*** **(** a **==** b **)** **{**  doX **();**  **}**  ***if*** **(** a **==** b **)** **{**  doW **();**  **}**  ***var*** j **=** 1 **;**  ***var*** k **=** 0 **;** |

#### Do not use the same property name when declaring an object

Javascript accepts the declaration of the same attribute, but when there are multiple properties with the same name, Javascript will only update the last declared attribute value and ignore the previously declared duplicate attribute.

Non-compliant code example

|  |
| --- |
| ***var*** data **=** **{**  "key" **:** "value" **,**  "1" **:** "value" **,**  "key" **:** "value" **,** // Noncompliant - duplicate of "key"  'key' **:** "value" **,** // Noncompliant - duplicate of "key"  key **:** "value" **,** // Noncompliant - duplicate of "key"  \u006bey **:** "value" **,** // Noncompliant - duplicate of "key"  "\u006bey" **:** "value" **,** // Noncompliant - duplicate of "key"  "\x6bey" **:** "value" **,** // Noncompliant - duplicate of "key"  1 **:** "value" // Noncompliant - duplicate of "1"  **}** |

Correct spelling:

|  |
| --- |
| ***var*** data **=** **{**  "key" **:** "value" **,**  "1" **:** "value" **,**  "key2" **:** "value" **,**  'key3' **:** "value" **,**  key4 **:** "value" **,**  \u006bey5 **:** "value" **,**  "\u006bey6" **:** "value" **,**  "\x6bey7" **:** "value" **,**  1b **:** "value"  **}** |

#### Conditions in "if/else if" or "switch…case" blocks do not overlap.

In "if/else if" or "switch…case" blocks, only the first branch with true condition is executed. Therefore, repeating a condition (often by mistake when copying/pasting) can cause serious errors that we do not anticipate.

Non-compliant code example

|  |
| --- |
| ***if*** **(** param **==** 1 **)**  openWindow **();**  ***else*** ***if*** **(** param **==** 2 **)**  closeWindow **();**  ***else*** ***if*** **(** param **==** 1 **)** // Already have this condition  moveWindowToTheBackground **();**  ***switch* (** i **)** **{**  ***case*** 1 **:**  //...  ***break* ;**  ***case*** 3 **:**  //...  ***break* ;**  ***case*** 1 **:** // Already have this case  //...  ***break* ;**  ***default* :**  // ...  ***break* ;**  **}** |

Correct spelling:

|  |
| --- |
| ***if*** **(** param **==** 1 **)**  openWindow **();**  ***else*** ***if*** **(** param **==** 2 **)**  closeWindow **();**  ***else*** ***if*** **(** param **==** 3 **)**  moveWindowToTheBackground **();**  ***switch* (** i **)** **{**  ***case*** 1 **:**  //...  ***break* ;**  ***case*** 3 **:**  //...  ***break* ;**  ***default* :**  // ...  ***break* ;**  **}** |

#### Use the "length" attribute to specify DOM elements

When performing a search for an object, the length property should be used to determine whether the object is found.

Non-compliant code example

|  |
| --- |
| ***if*** **(** $ **(** "div.foo" **)** **)** **{** // Always return true even though no object is found  **}** |

Correct spelling:

|  |
| --- |
| // Testing whether a selection contains elements.  ***if*** **(** $ **(** "div.foo" **).** length **>** 0 **)** **{**  // this code only runs if elements were found  // ...  **}** |

#### Don't use the global "this" variable

When using the outermost declared global "this" variable, javascript will interpret it as a reference to the window object. If you want to declare a global variable, removing the this variable still gives the same result.

Non-compliant code example

|  |
| --- |
| ***this* .** foo **=** 1 **;** // Do not obey  console **.** log **( *this* .** foo **);** // Do not obey  ***function*** MyObj **()** **{**  ***this* .** foo **=** 1 **;** // Compliance  **}**  MyObj **.** func1 **=** ***functions* ()** **{**  ***if*** **( *this*** .foo **== \_** 1 **)** **{** // Compliance  // ...  **}**  **}** |

Correct spelling:

|  |
| --- |
| foo **=** 1 **;**  console **.** log **(** foo **);**  ***function*** MyObj **()** **{**  ***this* .** foo **=** 1 **;**  **}**  MyObj **.** func1 **=** ***functions* ()** **{**  ***if*** **( *this*** .foo **== \_** 1 **)** **{**  // ...  **}**  **}** |

#### Model attribute names without spaces " "

When using the Backbone.js framework, model property names should not contain spaces because the Events object accepts a space-defined event list. Therefore, an attribute name containing spaces can be misinterpreted.

Non-compliant code example

|  |
| --- |
| Person **=** Backbone **.** Model **.** extend **({**  defaults **:** **{**  'first name' **:** 'Bob' **,** // Do not obey  'birth date' **:** ***newDate*** ( **)** // Do not obey  **},**  **});** |

Correct spelling:

|  |
| --- |
| Person **=** Backbone **.** Model **.** extend **({**  defaults **:** **{**  firstName **:** 'Bob' **,**  birthDate **:** ***newDate*** ( **)**  **},**  **});** |

#### Operators "+" and "-" are not used with Object

The + and - operators are used to convert the value type to a numeric value, but not every value can be converted to Number, then the result always returns NaN.

Non-compliant code example

|  |
| --- |
| ***var*** obj **=** **{** x **:** 1 **};**  doSomethingWithNumber **(+** obj **);** // Not according to the law  ***function*** foo **(){**  ***return*** 1 **;**  **}**  doSomethingWithNumber **(-** foo **);** //Not according to the law |

Correct spelling:

|  |
| --- |
| ***var*** obj **=** **{** x **:** 1 **};**  doSomethingWithNumber **(+** obj **.** x **);**  ***function*** foo **(){**  ***return*** 1 **;**  **}**  doSomethingWithNumber **(-** foo **());**  ***var*** str **=** '42' **;**  doSomethingWithNumber **(+** str **);**  //Exception  // Operators +, - can be used with objects of primitive type.  ***var*** b **=** ***new*** Boolean **( *true* );**  doSomethingWithNumber **(-** b **);** // Compliant |

#### In the "password" input tag set the "autocomplete" attribute to "off"

Most browsers automatically populate the 'password' input tag when the password has been previously entered. This can cause information insecurity errors. In HTML this can be fixed by setting the autocomplete attribute for this tag to off.

Non-compliant code example

|  |
| --- |
| **HTML5:**  <input type = **"password"** /> |

Correct spelling:

|  |
| --- |
| **HTML5:**  <input type = **"password"** autocomplete= **"off"** /> |

# APPENDIX 0 4: PROGRAMMING GUIDELINES TO AVOID POTENTIAL ERRORS PHP APPLICATION CODE

#### Don't use $this in a static context

$this refers to the current instance of the class, however the static method can be executed without instantiating the class, so using $this in a static context can result in runtime errors (runtime). error).

Non-compliant code example

|  |
| --- |
| class Clazz **{**  $name **=** NULL **;** // instance variable  public static function foo **(){**  **if** **(** $this **->** name **!=** NULL **)** **{**  // ...  **}**  **}**  **}** |

Correct spelling:

|  |
| --- |
| class Clazz **{**  $name **=** NULL **;** // instance variable  public static function foo **(** $nameParam **){**  **if** **(** $nameParam **!=** NULL **)** **{**  // ...  **}**  **}**  **}** |

#### Do not use exit(...), die(...)

Do not use exit(...), die(...) statements in PHP web pages as this will cause bad user experience. Users may even think that the web site is down or has a serious error.

Non-compliant code example

|  |
| --- |
| class Foo **{**  public function bar **(** $param **)** **{**  **if** **(** $param **===** 42 **)** **{**  exit **(** 23 **);**  **}**  **}**  **}** |

Correct spelling:

|  |
| --- |
| class Foo **{**  public function bar **(** $param **)** **{**  **if** **(** $param **===** 42 **)** **{**  **throw** **new** Exception **(** 'Value 42 is not expected.' **);**  **}**  **}**  **}**  //... |

#### Do not define Functions, Variables outside of Class

Defining Function, Variable global can cause the following risks:

* Name conflicts between objects
* It is very difficult to test a class that uses global functions
* Global variables can be updated from anywhere and can therefore have unexpected values

For processing, it is possible to transfer Functions and Variables into a certain Class and convert to static, then can be used without instantiating the Class.

Non-compliant code example

|  |
| --- |
| **<?** php  $name **=** "Bob" **;** // Noncompliant  function doSomething **(** $arg **)** **{** // Noncompliant  //...  **}**  class MyClass **{**  //...  **}** |

Correct spelling:

|  |
| --- |
| **<?** php  class MyClass **{**  public static function doSomething **(** $arg **)** **{** // Compliant  //...  **}**  //...  **}** |

#### Don't use the variable Variable variables

Using dynamically named variables makes the code confusing and difficult to maintain.

Non-compliant code example

|  |
| --- |
| $var **=** 'foo' **;**  $$var **=** 'bar' **;** //Noncompliant  $$$var **=** 'hello' **;** //Noncompliant  echo $foo **;** //will display 'bar'  echo $bar **;** //will display 'hello' |

#### No hard-coded username/password

Strings can be easily extracted by decompiling the program so that authentication information such as username/password must not be hard-coded.

Credentials need to be stored in encrypted configuration files or in a database.

Non-compliant code example

|  |
| --- |
| $uname **=** "steve" **;**  $password **=** "blue" **;**  connect **(** $uname **,** $password **);** |

Correct spelling:

|  |
| --- |
| $uname **=** getEncryptedUser **();**  $password **=** getEncryptedPass **();**  connect **(** $uname **,** $password **);** |

#### Don't use "global"

Using global variables is a bad habit, reducing reusability and making maintenance difficult. Variables should be passed as arguments to functions.

Non-compliant code example

|  |
| --- |
| $myGlobalVariable **;**  function foo **()**  **{**  global $myGlobalVariable **;** // Noncompliant  $GLOBALS **[** 'myGlobalVariable' **];** // Noncompliant  // ...  **}** |

Correct spelling:

|  |
| --- |
| function foo **(** $myStateVariable **)**  **{**  // ...  **}** |

#### No dynamic code execution

The eval function is a way to execute any piece of code while the program is running without compiling. According to the official PHP documentation, the implementation of eval() is dangerous and prone to unusual errors.

Non-compliant code example

|  |
| --- |
| eval **(** $code\_to\_be\_dynamically\_executed **)** |

#### Do not use the same operator on both sides of a binary expression

This is usually due to a mistake (copy/paste) or simply redundant code

This rule does not apply to the \*, +, = operators.

Non-compliant code example

|  |
| --- |
| **if** **(** $a **==** $a **)** **{** // always true  doZ **();**  **}**  **if** **(** $a **!=** $a **)** **{** // always false  doY **();**  **}**  **if** **(** $a **==** $b **&&** $a **==** $b **)** **{** // if the first one is true, the second one is too  doX **();**  **}**  **if** **(** $a **==** $b **||** $a **==** $b **)** **{** // if the first one is true, the second one is too  doW **();**  **}**  $j **=** 5 **/** 5 **;** //always 1  $k **=** 5 **-** 5 **;** //always 0 |

Exception

|  |
| --- |
| Left **-** shifting 1 onto 1 is common in the construction of bit masks **,** and is ignored **.**  $i **=** first **<<** 1 **;** // Compliant  $j **=** $a **<<** $a **;** // Noncompliant |

#### Do not declare other statements after Jump

Jump statements such as return, break, continue, goto, throw will interrupt the execution flow of the current block of code, the statements that follow in the block will be redundant and not executed. In some rare cases the code will still be executed after that but it will be very confusing and easy to confuse when it needs to be maintained.

Non-compliant code example

|  |
| --- |
| function fun **(** $a **)** **{**  $i **=** 10 **;**  **return** $i **+** $a **;**  $i **++;** // this is never executed  **}**  function foo **(** $a **)** **{**  **if** **(** $a **==** 5 **)** **{**  **goto** error **;**  **}** **else** **{**  // by the job  **}**  **return ;**  error **:**  printf **(** "don't use 5" **);** // this is reachable but unreadable  **}** |

Correct spelling:

|  |
| --- |
| function fun **(** $a **)** **{**  $i **=** 10 **;**  **return** $i **+** $a **;**  **}**  function foo **(** $a **)** **{**  **if** **(** $a **==** 5 **)** **{**  handleError **();**  **}** **else** **{**  // by the job  **}**  **return ;**  **}** |

#### Do not create redundant objects

There is no good reason for creating the object without use, in most cases this can be a sign of code deletion by mistake. Even if it wasn't by mistake, not causing an error would waste resources unnecessarily.

Non-compliant code example

|  |
| --- |
| **if** **(** $x **<** 0 **)** **{**  **new** foo **;** // Noncompliant  **}**  Compliant Solution  $var **=** NULL **;**  **if** **(** $x **<** 0 **)** **{**  $var **=** **new** foo **;**  **}** |

Correct example:

|  |
| --- |
| $var **=** NULL **;**  **if** **(** $x **<** 0 **)** **{**  $var **=** **new** foo **;**  **}** |

#### Do not use repeated conditions in the same "if/else if" and "cases" expressions

The switch expression or sequence of if/else if conditions is checked from the top down, and only the first branch that satisfies the condition true will be executed.

Repeating the conditional will result in some code never being executed. This is usually a copy/paste error and is very confusing for maintenance, eg:

* During maintenance, it is possible to mistakenly update the redundant code (the code is duplicated and never executed) instead of updating the code that needs to be fixed.
* In case the swith expression has a repeated case, if the first case is not terminated by break then the repeated case can continue to be executed and cause abnormal behavior.

Non-compliant code example

|  |
| --- |
| **if** **(** $param **==** 1 **)**  openWindow **();**  **else** **if** **(** $param **==** 2 **)**  closeWindow **();**  **else** **if** **(** $param **==** 1 **)** // Noncompliant  moveWindowToTheBackground **();**  **switch (** $i **)** **{**  **case** 1 **:**  //...  **break ;**  **case** 3 **:**  //...  **break ;**  **case** 1 **:** // Noncompliant  //...  **break ;**  **default :**  // ...  **break ;**  **}** |

Correct spelling:

|  |
| --- |
| **if** **(** $param **==** 1 **)**  openWindow **();**  **else** **if** **(** $param **==** 2 **)**  closeWindow **();**  **else** **if** **(** $param **==** 3 **)**  moveWindowToTheBackground **();**  **switch (** $i **)** **{**  **case** 1 **:**  //...  **break ;**  **case** 3 **:**  //...  **break ;**  **default :**  // ...  **break ;**  **}** |

#### The logical condition needs to be guaranteed to not access the Null object.

When writing logical conditions, we need to pay attention to which condition is executed first, which condition is executed later to ensure that there are no cases of accessing the Null object.

Non-compliant code example

|  |
| --- |
| **if** **(** $obj **==** **null** **&&** $obj **->** isOpen **())** **{**  echo "Object is open" **;**  **}**  **if** **(** $obj **!=** **null** **||** $obj **->** isOpen **())** **{**  echo "Object is not open" **;**  **}** |

Correct spelling:

|  |
| --- |
| **if** **(** $obj **==** **null** **||** $obj **->** isOpen **())** **{**  echo "Object is open" **;**  **}**  **if** **(** $obj **!=** **null** **&&** **!** $obj **->** isOpen **())** **{**  echo "Object is not open" **;**  **}** |

#### Use break at the end of Switch cases . expressions

When execution is not explicitly stopped at the end of the switch case expression, the next case will continue to be executed. This may be intentional by the programmer in some cases, but in most cases it is due to forgetting and can cause unusual handling.

Non-compliant code example

|  |
| --- |
| **switch** **(** $myVariable **)** **{**  **case** 1 **:**  foo **();**  **break ;**  **case** 2 **:** // Both 'doSomething()' and 'doSomethingElse()' will be executed. Is it on purpose ?  do\_something **();**  **default :**  do\_something\_else **();**  **break ;**  **}** |

Correct spelling:

|  |
| --- |
| **switch** **(** $myVariable **)** **{**  **case** 1 **:**  foo **();**  **break ;**  **case** 2 **:**  do\_something **();**  **break ;**  **default :**  do\_something\_else **();**  **break ;**  **}** |

Exception:

|  |
| --- |
| This rule is relaxed in the following cases **:**  **switch** **(** $myVariable **)** **{**  **case** 0 **:** // Empty case used to specify the same behavior for a group of cases.  **case** 1 **:**  do\_something **();**  **break ;**  **case** 2 **:** // Use of continue statement  **continue ;**  **case** 3 **:** // Case includes a jump statement (exit, return, break &etc)  exit **(** 0 **);**  **case** 4 **:**  echo 'Second case, which falls through' **;**  // no break <- comment is used when fall-through is intentional in a non-empty case body  **default :** // For the last case, use of break statement is optional  doSomethingElse **();**  **}** |