Sonar Documentation

1. Constant names should comply with a naming convention

Shared coding conventions allow teams to collaborate efficiently. This rule checks that all constant names match a provided regular expression.

Noncompliant Code Example

With the default regular expression `^[A-Z][A-Z0-9]*(_[A-Z0-9]+)*$`:

```
public class MyClass {
  public static final int first = 1;
}

public enum MyEnum {
  first;
}
```

Compliant Solution

```
public class MyClass {
  public static final int FIRST = 1;
}

public enum MyEnum {
  FIRST;
}
```

Old Code -

```
public enum DatabaseEnum {

	in_memory, sql;
}
```

New Code -

```
public enum DatabaseEnum {
	IN_MEMORY, SQL;
}
```

2. "switch" statements should have at least 3 "case" clauses

`switch` statements are useful when there are many different cases depending on the value of the same expression.

For just one or two cases however, the code will be more readable with `if` statements.

Noncompliant Code Example

```
switch (variable) {
  case 0:
    doSomething();
    break;
  default:
    doSomethingElse();
    break;
}
```

Compliant Solution

```
if (variable == 0) {
  doSomething();
} else {
  doSomethingElse();
}
```

Old Code -

```
public static BaseDao<Product> getInstance(DatabaseEnum type) {
	switch(type){
		case IN_MEMORY:
			baseProductDao = InMemoryProductDao.getInstance();
			break;
		case SQL:
			baseProductDao = null;
			break;
	}
	return baseProductDao;
}
```

New Code -

```
public static BaseDao<Product> getInstance(DatabaseEnum type) {

	if(type == DatabaseEnum.IN_MEMORY) {
		baseProductDao = InMemoryProductDao.getInstance();

	} else if(type == DatabaseEnum.SQL) {
		baseProductDao = null;
	}

	return baseProductDao;
}
```

3. Redundant casts should not be used

Unnecessary casting expressions make the code harder to read and understand.

Noncompliant Code Example

```
public void example() {
  for (Foo obj : (List<Foo>) getFoos()) {  // Noncompliant; cast unnecessary
because List<Foo> is what's returned
    //...
  }
}

public List<Foo> getFoos() {
  return this.foos;
}
```

Compliant Solution

```
public void example() {
  for (Foo obj : getFoos()) {
    //...
  }
}

public List<Foo> getFoos() {
  return this.foos;
}
```

Old Code -

```
private BaseDao<Product> productDao =
(InMemoryProductDao)ProductDaoFactory.getInstance(DatabaseEnum.IN_MEMORY);
```

New Code -

```
private BaseDao<Product> productDao =
ProductDaoFactory.getInstance(DatabaseEnum.IN_MEMORY);
```

4. Standard outputs should not be used directly to log anything

When logging a message there are several important requirements which must be fulfilled:

- The user must be able to easily retrieve the logs
- The format of all logged message must be uniform to allow the user to easily read the log
- Logged data must actually be recorded
- Sensitive data must only be logged securely

If a program directly writes to the standard outputs, there is absolutely no way to comply with those requirements. That's why defining and using a dedicated logger is highly recommended.

Noncompliant Code Example

```
System.out.println("My Message");  // Noncompliant
```

Compliant Solution

```
logger.log("My Message");
```

Old Code -

```
public class Main {

   public static void main(String[] args) {
      System.out.print("Hello ");

      System.out.print("World!");
   }
}
```

New Code -

```
import java.util.logging.*;

public class Main {

       private static Logger logger = Logger.getLogger(Main.class.getName());

   public static void main(String[] args) {
      logger.info("Hello ");

      logger.info("World!");
   }
}
```

5. Array designators "[]" should be on the type, not the variable

Array designators should always be located on the type for better code readability. Otherwise, developers must look both at the type and the variable name to know whether or not a variable is an array.

Noncompliant Code Example

```
int matrix[][];   // Noncompliant
int[] matrix[];   // Noncompliant
```

Compliant Solution

```
int[][] matrix;   // Compliant
```

Old Code -

```
public static void main(String args[])
```

New Code -

```
public static void main(String[] args)
```