Common Code Violations in Java

Published at DZone with permission of Veera Sundar, author and DZone MVB
http://java.dzone.com/articles/common-code-violations-java

Format source code and Organize imports

Eclipse provides the option to auto-format the source code and organize the imports (thereby removing unused ones). You can use the following shortcut keys to invoke these functions.

- **Ctrl + Shift + F** – Formats the source code.
- **Ctrl + Shift + O** – Organizes the imports and removes the unused ones.

Instead of you manually invoking these two functions, you can tell Eclipse to auto-format and auto-organize whenever you save a file. To do this, in Eclipse, go to **Window -> Preferences -> Java -> Editor -> Save Actions** and then enable **Perform the selected actions on save** and check **Format source code** + **Organize imports**.

Avoid multiple returns (exit points) in methods

In your methods, make sure that you have only **one exit point**. Do not use returns in more than one places in a method body.

For example, the below code is **NOT RECOMMENDED** because it has more then one exit points (return statements).

```
private boolean isEligible(int age){
  if(age > 18){
    return true;
  }else{
    return false;
  }
}
```

The above code can be rewritten like this (of course, the below code can be still improved, but that'll be later).

```
private boolean isEligible(int age){
  boolean result;
  if(age > 18){
    result = true;
  }else{
    result = false;
  }
  return result;
}
```

Simplify if-else methods

We write several utility methods that takes a parameter, checks for some conditions and returns a value based on the condition. For example, consider the **isEligible** method that you just saw in the previous point.

```
private boolean isEligible(int age){
  boolean result;
  if(age > 18){
    result = true;
  }else{
    result = false;
  }
  return result;
}
```

The entire method can be re-written as a **single return** statement as below.

```
private boolean isEligible(int age){
  return age > 18;
}
```

Do not create new instances of Boolean, Integer or String

Avoid creating new instances of Boolean, Integer, String etc.

For example, instead of using **new Boolean(true)**, use **Boolean.valueOf(true)**.

The later statement has the same effect of the former one but it has improved performance

Use curly braces around block statements

Never forget to use curly braces around block level statements such as ***if***, ***for***, ***while***. This reduces the ambiguity of your code and avoids the chances of introducing a new bug when you modify the block level statement.

NOT RECOMMENDED

```
if(age > 18)
  return true;
else
  return false;
```

RECOMMENDED

```
if(age > 18){
  return true;
}else{
  return false;
}
```

Mark method parameters as final, wherever applicable

Always mark the method parameters as ***final*** wherever applicable. If you do so, when you accidentally modify the value of the parameter, you'll get a compiler warning. Also, **it makes the compiler to optimize the byte code in a better way**.

RECOMMENDED

```
private boolean isEligible(final int age){ ... }
```

Name *public static final* fields in UPPERCASE

Always name the *public static final* fields (also known as *Constants*) in **UPPERCASE**. This lets you to easily differentiate constant fields from the local variables.

NOT RECOMMENDED

```
public static final String testAccountNo = "12345678";
```

RECOMMENDED

```
public static final String TEST_ACCOUNT_NO = "12345678";,
```

Combine multiple *if* statements into one

Wherever possible, try to combine multiple *if* statements into single one.

For example, the below code;

```
if(age > 18){
  if( voted == false){
    // eligible to vote.
  }
}
```

can be combined into single *if* statements, as:

```
if(age > 18 && !voted){
  // eligible to vote
}
```

switch should have *default*

Always add a *default* case for the *switch* statements.

Avoid duplicate string literals, instead create a constant

If you have to use a string in several places, avoid using it as a literal. Instead create a String constant and use it.

For example, from the below code:

```
private void someMethod(){
  logger.log("My Application" + e);
  ....
  ....
  logger.log("My Application" + f);
}
```

The string literal “My Application” can be made as an Constant and used in the code.

```
public static final String MY_APP = "My Application";

private void someMethod(){
  logger.log(MY_APP + e);
  ....
  ....
  logger.log(MY_APP + f);
}
```

Additional Resources

- A collection of Java best practices.
- List of available Checkstyle checks.
- List of PMD Rule sets

Published at DZone with permission of Veera Sundar, author and DZone MVB.

Articulo Original: http://java.dzone.com/articles/common-code-violations-java