Introduction to Coding Conventions

Patrick Schäfer
patrick.schaefer@hu-berlin.de

Semesterprojekt: Implementierung eines Brettspiels, WS 18/19
Good or Bad Coding Style?

```java
int i = 0;
while (i < 10)
    System.out.println(i);
    i++;
```

• What is the expected output?
Good or Bad Coding Style?

```java
int i = 0;
for (i = 0; i < 10; i++) {
    System.out.println(i);
}
```

- What is the expected output?
HashMap<String, Movie> actorsMap = new HashMap<>();
String actor = "";
if (actorsMap.containsKey(actor)) {
    actorsMap.get(actor).movieCount++;
} else {
    actorsMap.put(actor, new Actor(actor));
}

• What is the functionality of this code?
Motivation

• Goal: Self-documenting code that is easy to read.

• A coding convention defines the style of your source code.

• A team should use the same standard practices for:
  • **naming** classes, variables or functions.
  • **commenting** and **formatting** (indentation and brackets).

• Different conventions reasonable for every language (JAVA; C#; C++) and team.
Indentation and Bracket Placement Examples

```java
if (actorsMap.containsKey(actor))
{
    actorsMap.get(actor).movieCount++;
}
else
{
    actorsMap.put(actor, new Actor(actor));
}

• ANSI C Style
```
Indentation and Bracket Placement Examples

```java
if (actorsMap.containsKey(actor)) {
    actorsMap.get(actor).movieCount++;
} else {
    actorsMap.put(actor, new Actor(actor));
}
```

• Kernighan and Ritchie Style
Benefits

• No need to **reformat** code and **rename** variables and methods whenever working on code written by **others**.

• Source code is much **easier to understand** when reasonably formatted:

```java
if (condition) {
    // statement;
    otherStatement;
}

if (condition) {
    statement;
    otherStatement;
}

if (condition) {
    statement;
    otherStatement;
}
```
Identifiers

• Upper Case with Underscores:
  THIS_IS_AN_EXAMPLE

• Lower Camel Case:
  thisIsAnExample

• Upper Camel Case:
  ThisIsAnExample

• Lower Case with Underscores:
  this_is_an_example
Typical Conventions

• Classes use Upper Camel Case
  • eg: MovieFactory

• Functions use lower Camel Case
  • eg: readMovie

• Variables use lower Camel Case
  • eg: movies
Indentation, Spaces and Tabs

• Don’t mix tabs and spaces!
• Best is to set your editor to replace tabs by 2-4 spaces when you enter a tab.
• Does (spaces or tabs):
  ```java
  if (condition) {
    block;
  } else {
    block;
  }
  ```
• Don’t (space + tabs):
  ```java
  if (condition) {
    block;
  } else {
    block;
  }
  ```
(Good) Examples for Coding Conventions

do
use PascalCasing for class names and method names.

```csharp
public class ClientActivity
{
    public void ClearStatistics()
    {
        //...
    }
    public void CalculateStatistics()
    {
        //...
    }
}
```

Why: consistent with the Microsoft's .NET Framework and easy to read.

(Good) Examples for Coding Conventions

**do** use *camelCasing* for method arguments and local variables.

```csharp
public class UserLog
{
    public void Add(LogEvent logEvent)
    {
        int itemCount = logEvent.Items.Count;
        // ...
    }
}
```

*Why:* consistent with the Microsoft's .NET Framework and easy to read.

(Good) Examples for Coding Conventions

**do not**

use *Hungarian* notation or any other type identification in identifiers

```csharp
1. // Correct
2. int counter;
3. string name;
4. 
5. // Avoid
6. int iCounter;
7. string strName;
```

*Why*: consistent with the Microsoft’s .NET Framework and Visual Studio IDE makes determining types very easy (via tooltips). In general you want to avoid type indicators in any identifier.
(Good) Examples for Coding Conventions

**do not** use **Screaming Caps** for constants or readonly variables

1. // Correct
   
   ```csharp
   public static const string ShippingType = "DropShip";
   ```

2. // Avoid
   
   ```csharp
   public static const string SHIPPINGTYPE = "DropShip";
   ```

*Why:* consistent with the Microsoft's .NET Framework. Caps grab too much attention.

(Good) Examples for Coding Conventions

```csharp
// Correct
UserGroup userGroup;
Assignment employeeAssignment;

// Avoid
UserGroup usrGrp;
Assignment empAssignment;

// Exceptions
CustomerId customerId;
XmlDocument xmlDoc;
FtpHelper ftpHelper;
UriPart uriPart;
```

Why: consistent with the Microsoft’s .NET Framework and prevents inconsistent abbreviations.
(Good) Examples for Coding Conventions

**do**

use implicit type `var` for local variable declarations. Exception: primitive types (int, string, double, etc) use predefined names.

1. `var stream = File.Create(path);`
2. `var customers = new Dictionary<>();`
3. `// Exceptions`
4. `int index = 100;`
5. `string timeSheet;`
6. `bool isCompleted;`

**Why:** removes clutter, particularly with complex generic types. Type is easily detected with Visual Studio tooltips.

(Good) Examples for Coding Conventions

```csharp
// Correct
class Program
{
    static void Main(string[] args)
    {
    }
}
```

Why: Microsoft has a different standard, but developers have overwhelmingly preferred vertically aligned brackets.

(Good) Examples for Coding Conventions

```
1. // Correct
2. public class Account
3. {
4.     public static string BankName;
5.     public static decimal Reserves;
6.     
7.     public string Number {get; set;}
8.     public DateTime DateOpened {get; set;}
9.     public DateTime DateClosed {get; set;}
10.    public decimal Balance {get; set;}
11. 
12.    // Constructor
13.    public Account()
14.    {
15.        // ...
16.    }
17. }
```

Why: generally accepted practice that prevents the need to hunt for variable declarations.

Conclusion

• Agree on a coding guideline within your team stick to it!

• Good starting points:
  • C#
    • http://www.dofactory.com/reference/csharp-coding-standards