CS3134 Java Style Guide

Linan Qiu

This is a short but important collection of coding standards that we expect you to follow for this class. You will find that adhering to this code results in readable, maintainable, and easy to debug code. Beyond this class, you will find that good code style is essential not only for yourself but also for any large codebase engineered by a team of developers.

We do not claim ownership over these tips. Rather, they are inspired by Google's Java Style Guide which we encourage you to spend 7 minutes reading. Multiple TAs over the generations (James Lin, Andrew Goldin, Madhavan Somanathan) have also contributed to this guide.

1 Source File Basics

1.1 File Name

Source file name consists of the name of the class plus the `.java` extension.

1.2 File Header

Include your name, UNI, and description of the file in each Java source file that you write. This allows you to quickly identify the author of the code and the code's purpose.

1.1 File Encoding

Source files can be either UTF-8 or ASCII.

```java
/* James Lin
 * jl3782
 * HanSolo.java - killed
 */

public class HanSolo {
  // ...
}
```

1.3 Use Default Package

Use the default package for all your classes. Since this course does not require packages, you usually end up creating classes within packages as a result of IDEs' over-helpfulness. Make sure you leave the package row empty when you create a new class in your IDE (screenshot attached for Eclipse) even though it warns you otherwise. While it may be a good idea to put your classes in packages for large codebases, you won't need them for this class and you will instead give your TA compilation troubles. **Do not create your class within a package.**

That means instead of declaring your class file as such:

```
1 package somepackage;
2
3 public class Bulbasaur {
4   // ...
5 }
```

You should only have

```
1 public class Bulbasaur {
2   // ...
3 }
```

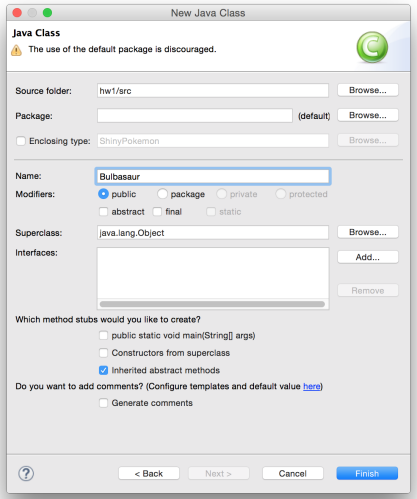

Figure 1: Creating a new class in Eclipse under the default package

2 Formatting

2.1 Brace Styles

You are required to choose between the K&R style or Allman style for creating blocks of code (e.g. in `if` blocks, `while` and `for` loops, defining classes etc). **Be consistent** throughout your code (ie. don't switch between styles in your code).

```
1  // K&R
2  for(int i = 0; i < array.length; i++) {
3    System.out.println(array[i]);
4  }
5
6  // Horstmann
7  for(int i = 0; i < array.length; i++)
8  {
9    System.out.println(array[i]);
10 }
```

2.2 Use Braces Wherever Optional

Braces are used with `if`, `else`, `for`, `do` and `while` statements, even when the body is empty or contains only a single statement.

```
// while this compiles
for (int i = 0; i < array.length; i++)
  System.out.println(array[i]);

// you should instead do this
for(int i = 0; i < array.length; i++) {
  System.out.println(array[i]);
}
```

2.3 Indentation

You can indent using **tabs, 2 spaces, or 4 spaces**. We allow tabs for convenience. However, in the industry, most companies (including Google) requires that tabs be translated into spaces. Most IDEs / text editors do this automatically should you set it up to do so.

```
// indentation using 2 spaces
int i = 0;
while (i < array.length) {
  double random = Math.random();
  if (random < 0.5) {
    array[i] = 0;
  } else {
    array[i] = i;
  }
}
```

2.4 Line Length

Keep your lines at **80 characters or under**. You will need to break up or wrap lines and comments that are longer than this. Again, most IDEs can do this for you.

```
String str = "This is a long strong " +
    "that needs to be broken up";
```

2.5 White Space

Place a blank line between each new thought. This allows your reader to follow your thought process step-by-step and keeps your code from getting cluttered. The code below demonstrates a program getting an object's data then displaying the information. The break shows the separation in thought even though the code is in a single method.

```
public boolean canDrive(Person person) {
  int age = person.getAge();

  return age >= 16;
}
```

3 Naming

3.1 Class Names

Class names are written in UpperCamelCase (e.g. `HanSolo` or `BinaryTree`) Class names are typically nouns or noun phrases. (e.g. `Character` or `ImmutableList`). Interface names may also be nouns or noun phrases (e.g. `List`), but may sometimes be adjectives or adjective phrases instead (for example, `Readable`).

3.1 Variable Names

Variable names (static or otherwise) are written in lowerCamelCase.

These names are typically nouns or noun phrases. For example, `computedValues` or `index`.

3.1 Method Names

Method names are written in lowerCamelCase (e.g. `kissLeia` or `toString`).

Method names are typically verbs or verb phrases. For example, `sendMessage` or `stop`.

3.2 Constant Names

Constant names use `CONSTANT_CASE`: all uppercase letters, with words separated by underscores.

```
1 public static final int ANSWER_TO_UNIVERSE = 42;
2 public static final String WARNING = "You shall not pass";
```

3.3 Meaningful Names

Give your classes, variables, and methods meaningful names. For example, instead of calling a class `Robot` (not informative, since all programs are robots), call it `Calculator`. Instead of calling a variable `number`, call it `height` if it is used to denote the height of a person.

3.4 Avoid Hardcoding Magic Numbers

Avoid hardcoding any constants that bear significance affecting the output of your program. If you have to hardcode them, give them meaningful names.

```
1  // Bad
2  public double weight(double mass) {
3    return mass * 9.81;
4  }
5
6  // Good
7  public static final double GRAVITY = 9.81;
8
9  public double weight(double mass) {
10   return mass * GRAVITY;
11 }
```