Java Coding Standard

This coding standard is based on Oracle's Java Coding Standard and is widely used by Java programmers. A standard helps make code easy to read & share.

*You **must use this standard** for all assignments in the OOP course to get full credit.*

Example	Explanation
/* * This source code is Copyright 2012 by Jim Brucker. */ **package** coinpurse;	*Optional* comment (not Javadoc) at start of file. This comment is for copyright or notes to other developers. The **package** for this class. Package name must be **lowercase**. **package** name must be same as **folder** name.
import java.util.List; **import** java.util.Scanner; **/**** * A Coin Purse with a fixed capacity, it * manages insert and withdraw of coins. * **@author** Your Name * **@version** 2018.01.15 ***/**	Import classes from other packages. Must come after "package" statement. **Javadoc** comment for the first class, begins with **/**** . First sentence should describe what the class does and end with a period. **Don't** write "This class..." (useless waste of words). Include these tags: **@author** and your name. **Don't use parenthesis**! @author another author -- use one tag for each author. **@version** is a version number or date modified. Version must **increase**, so use year.mon.day eg 2012.01.15
public class Purse **implements** Comparable	Class name should begin with **capital letter** and use mixed case, as shown. All uppercase name is allowed **only if** name is an acronym, such as URL.

/** convert nanoseconds to seconds */
static final double NANOSECOND = 1.0E-9;
static final long MAX_SIZE = 1000;
	Declare **constants** first.
	Constant names should be **UPPERCASE** with words separated by _ (underscore).
	Public constants should have a Javadoc comment.

/** The next available id number */ private static int nextId = 1;	Declare **static variables** after constants. Static attributes are unique to the class (not each object).

// birthday is final because it
// should not change.
private **final** Date **birthday**;
	If **final** is used simply to prevent reassignment of a reference, rather than a constant *value* that has special meaning, then **use camel-case, just like ordinary variable name**.
	"final" is often used for attributes and local variables we don't want to change after the first assignment.

/** Number of items purse can hold. */ private int capacity; /** List of items in the purse. */ private List<Coin> coins;	**Declare (object) attributes next.** You should declare the access level (public, private, or protected); usually **private**. Attribute **names** should be **camelCase**, beginning with a lowercase letter. Write a Javadoc comment if the meaning of attribute is not obvious. Comment should come before the attribute declaration.
private String productCode; private Money total; /* **Bad variable names** */ private String prodCode; private Money t; private int n; private double Total;	**Good names:** descriptive, camel case (first letter is lowercase, each other words start with uppercase) **bad:** don't use abbreviations **bad:** names like "t" and "n" are not descriptive **wrong:** variable names should begin with lowercase
/** Initialize a new purse. * **@param** size is the capacity of purse */ public Purse(int size) { ...	**Constructors** should have Javadoc comment. **@param** tag describes each parameter. No space between class name and "(". A Constructor does not have a return value -- not even void.

```
/**
 * Compare coins by value.
 * @param coin is a Coin to compare to this.
 * @return -1 if this coin has lower value, ...
 * @throws NullPointerException if coin is null
 * @see java.lang.Comparable#compareTo(Object)
 */
public int compareTo(Coin coin) {
   body of method
}
```

Methods: Write a Javadoc comment before every method, except for trivial get and set methods.

1. First sentence of comment should describe what the method does. Write a complete sentence, ending with period.
2. **Don't write:** "This method does..." (waste of words).
3. Include javadoc **tags** for:

`@param` parameter description of parameter

`@return` describe the return value, if any

`@throws` list any exceptions thrown

`@see` (optional) other methods containing related documentation

```
public boolean isFull( )
{
    return count() >= this.capacity;
}
```

Method name should be camelCase (lowercase first letter)

Method "{ ... }" block: *Two ways to format*.

You can put left brace "**{**" on **same line** as method name (as in compareTo example) or on a **separate line** (this example).

```
while (count < MAX_COUNT) {
    if (count%10 == 0) {
        doReport();
        print(count);
    }
    else {
        doSomethingElse();
    }
    count++;
}
```

Indent blocks consistently!

Indent blocks using 1 tab. Set tab size = 4 spaces.

Use TAB to indent, not spaces.

Code inside block should be at same indent.

Netbeans: use Options > Editor > Formatting and UNSELECT "Expand tabs to spaces".

Eclipse: TAB is the default for indentation.

```
if (amount <= 0) {
     System.out.println("Invalid amt");
     return;
}
else deposit(amount);
```

"if" blocks { ... }

Indent the block inside { ... } as shown here.

When "then" or "else" clause is just one statement, you can omit the { } as in this example.

```
if (size < 0) size = 1;
while (count-- > 0) readLine( );
```

Use space outside of "(...)" in "`if (...)`" and "`while (...)`".

```
double length = Math.hypot(2, 3);
Date now = new Date( );
```

No space between method name and "(".

No space after class name in "new Xxx( )"

```
int total = quantity * unitPrice;
double descriminant = b*b - 4*a*c;
```

`Use space` around =, >, <, and arithmetic operators. For long operations you can omit space around * and /.

```
public Class Purse {
     public int getTotal() {
          while ( coins.hasNext() ) {
```

`Space` before left brace "{" when on same line as class or method name.

```
public void addToCount() {
     count++;
```

`NO Space` between method name and "**(**".

`NO Space` between variable name and **++** or **--**.

```
public static void main(String[] args) {
     Game game =  new Game( );
     ScoreBoard scoreboard =
               new ScoreBoard(game);
     game.play( );
}
```

Use **`main`** to initialize the program, not for program logic!

The program's logic should be in methods, not in the main method.

Usually main creates objects, connects objects together, and then invokes some method to "run" the application.

```
long now = System.nanoTime( );
double elapsed = (now - start)*1.0E+9;
// what does 1.0E+9 mean?
```

`Don't use literal values` for values that have special meaning in your code.

It is hard to understand and hard to modify.

`final double NANOS_PER_SECOND = 1.0E+9;`
`long now = System.nanoTime( );`
`double elapsed =`
`(now – start)*NANOS_PER_SECOND;`
	Use Named Constants for things that have special meaning in your code.
	UPPERCASE for names of constants (final values).