Java Coding Standard

This coding standard is based on Sun's Java Coding Standard and the experience of many programmers. It is designed to help make code easy to read & share.

*You **must use this standard** for all assignments in the OOP course. Points are deducted for not adhering to it.*

Source File Structure	Explanation
/* * This source code is Copyright 2012 by Jim Brucker. */ **package** coinpurse; **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** 2012.07.15 ***/**	*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 classes from other packages. Must come after "package" statement. import is like C# "using". **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 names 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 or ISBN.

/** 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.

// birthday is final because it
// should not change.
private **final** Date **birthday**;
public Person(String name, Date bday) {
   this.birthday = bday;
	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. For example, a Person's birthday should not change.

/** Scanner for input from console */
   private static Scanner console
                 = new Scanner(...);
   private static int nextID = 1;
	Declare **static variables** second (after constants).
	This Scanner doesn't *really* belong in Purse class, it is just an example.

/** Number of items purse can hold. */
   private int capacity;
   /** List of items in the purse. */
   private List<Coin> coins;
	Declare attributes next. You must declare the access level (**public**, **private**, or **protected**); usually it is **private**.
	Attribute **names** should be **camelCase**, beginning with a lowercase letter.
	Write a Javadoc comment if the meaning of attribute is not obvious. Javadoc comment should come before the attribute declaration.

```
private String productCode;
private Money total;
/* Bad 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 names:

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.

A Constructor does not have a return value -- not even void.

No space between class name and "(".

```
/**
 * 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/set methods.

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

```
@param parameter descriptions
@return describe the return value, if any
@throws list any exceptions thrown
@see (optional) other methods containing
related documentation
```

```
public int compareTo(Coin coin)
{
    body of method
}
```

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

You can put left brace "**{**" on **__same__ line** as method name (as in previous example) or on a **__separate__ line** (this example). Be consistent.

```
/** Get capacity of the purse.
 *  @return number of coins it can hold
 */
public int getCapacity( ) {
    return capacity;
}
```

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

Use TAB __not__ spaces to indent.
In Netbeans, use Options > Editor > Formatting and UNSELECT "Expand tabs to spaces".

In Eclipse TABs are the default.

BlueJ automatically converts TAB to spaces.

A simple *accessor* (*getter*) method.

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

Indent blocks consistently!

Code inside block should be at same indent.

Closing "}" must match previous indent level.

Use TAB for indent, not spaces.

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

"if" blocks:

When "then" or "else" clause contains more than one statement, indent as in previous example.

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

```
if (size < 0) size = 1;
while (count-- > 0) readLine( );
// no space before "(" in these cases:
double diag = Math.hypot(2, 3);
Date now = new Date( );
```

Use space before "(" and after ")" in "if" and "while".

Exceptions: no space between method name and "(" for parameters. No space after class name and "(" in new.

`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 **`--`**.

`long now = System.nanoTime( );`
`double elapsedTime =`
`    (now  - startTime)*1.0E+9; // `***`what?`***
	`Don't use literal values` for values that have special meaning in your code.
	It is hard to understand and modify.
	In this example, what is meaning of `1.0E+9` ?

`final double NANOSEC_PER_SECOND = 1.0E+9;`
`long now = System.nanoTime( );`
`double elapsedTime =`
`  (now  - startTime)*NANOSEC_PER_SECOND;`
`  // `***`better`***
	Use Named Constants for things that have special meaning in your code.
	UPPERCASE for names of constants (final values).

`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, but not the main method.
	Usually main creates objects, connects objects together, and then invokes some method to "run" the application.