Why Do All Cars Have Cigarette Lighters?

a) Because most people smoke.

b) Because the cigarette lighter is actually the most efficient and robust design for optimal delivery of automobile battery power to a great variety of electronic devices.

c) Because the cigarette companies have powerful lobbyists.

d) Because the Chinese want good Americans to get cancer.

e) Because it is a standard in a sea of chaos.
Order vs Chaos: Standard Since 1925

Car Cigarette Lighter Charger for a few Cell Phone Models

....But, what about my iPod, my Dell laptop, my wife’s Sony laptop, my Nintendo DS, my new Norelco razor and my old Norelco razor?

wikipedia: However, they were not originally designed to provide electrical power for miscellaneous devices, and are not an ideal power connector for several reasons:......

---

CS-152 Code Standard

- All CS-152 assignments must follow the great and hallowed CS-152 code standard.
- This standard does not necessarily represent the best nor the only good way to write Java code.
- If you have experience programming, then these standards may not be the standards you are used to using.
- However, in this class, these are the standards we will use.
Primary Reasons for Defined Standard

1. A standard makes it easier for the instructors to read your code.

2. A class standard makes it easier for a grader to recognize when a program does not use a consistent standard.
   Often when each student is allowed to define his or her own standard, students switch standards multiple times in a single project. It is tedious for a grader to deduce each person’s standard and then check for self-consistency.

3. Learning to adhere consistently to a coding standard is a good practice.

Coding Standard: Naming

- All variable names (fields) not declared final, shall begin with a lower case letter.
- All variables that do not ever change value shall be declared final and shall be all uppercase.
- All class variables (non-local variables) will be given descriptive names.
- Local variables will not be named O or I.
- All methods will be given descriptive names.
- All class names shall begin with an uppercase letter.
Coding Standard – Open Brackets

Open brackets must be the first non-space character on a line.

<table>
<thead>
<tr>
<th>ok</th>
<th>public class Hello</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>{</td>
</tr>
<tr>
<td></td>
<td>public static void main(String[] a)</td>
</tr>
<tr>
<td></td>
<td>{</td>
</tr>
<tr>
<td></td>
<td>System.out.println(&quot;Hello World&quot;);</td>
</tr>
<tr>
<td></td>
<td>}</td>
</tr>
<tr>
<td></td>
<td>}</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Not CS-152 standard</th>
<th>public class Hello</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>{</td>
</tr>
<tr>
<td></td>
<td>public static void main(String[] a)</td>
</tr>
<tr>
<td></td>
<td>{</td>
</tr>
<tr>
<td></td>
<td>System.out.println(&quot;Hello World&quot;);</td>
</tr>
<tr>
<td></td>
<td>}</td>
</tr>
</tbody>
</table>

Coding Standard – Closing Brackets

Closing brackets will be indented on a line with no other commands. The only exception being comments placed on the line with a closing bracket.

```java
if (x == 5) {
    y=y+1;
} //Comment here ok
else if (x == 7) {
    y=y+2;
}
if (x == 5) {
    y=y+1;
} else if (x == 7) {
    y=y+2;
}
```
Coding Standard – Blocks and {} 

- Whenever a structure spans more than one line, brackets must be used. For example:

  | ok       | if (x == 5) y=y+1; |
  | ok       | if (x == 5)         |
  |          | { y=y+1; }         |
  | Not CS-152 standard | if (x == 5) |
  |          | y=y+1;             |

Indenting

- Code blocks will be indented to show the block structure with **two spaces** per level.
- Tab characters shall **not** be used for indenting.
- All statements within a block must be indented to the same level.
Class Comments

At the top of every class file, there must be a comment block with the following information. Format the information as you think best.

('').***********************************
//Your first and last name
//
//Description of what the class
//is used for and how to use it.
//***********************************

Method Comments

At the top of every method, there must be a comment block with the following information. Format the information as you think best.

//////////////////////////////////////////////////
//Each parameter's type and name:
//input and/or output,
//its meaning,
//its range of values.
//Method's return value.
//Description of what the method does.
//Method's Algorithm
//////////////////////////////////////////////////
Within Method Comments

Whenever you have code that you think either you or your lab instructor might not understand in a quick look, add some helpful comments.

```java
double radius; //the radius of a circle

double radius; //in inches
```

80 Character Line Max

No line shall be more than 80 characters.

The best way to avoid overly long statements is by not doing too much in a single statement.

```java
int volume1 = getVolume(length1, width1, height1);
int volume2 = getVolume(length2, width2, height2);
if (volume1 > volume2)
    System.out.println("box 1 is bigger");
else
    System.out.println("box 2 is bigger");
```
Fixing Too Long a Line Example 2

- Another case where a temporary variable can shorten a line and improve readability.
- Creating the temporary variable \( c \) also improves code maintenance:

If the code changes so that the comparison needs to check \( \text{stack[topOfStack]} \) or \( \text{stack[topOfStack-2]} \), then Line 2 and 3 require only a single change while line 1 requires 4 changes.

```java
    //WRONG: 4 uses of stack[topOfStack-1] make the line too long.
2  char c = stack[topOfStack - 1];
3  if (c == '*' || c == '+' || c == '-' || c == '/')
    //Correct
```

Fixing Too Long a Line Example 3

- There are times when breaking a long statement into multiple statements is more awkward than keeping the long statement.
- In such cases, the statement should be broken in a logical place and each line over which the long statement is continued must be indented.
- The indenting must be at least 2 spaces, but can be more spaces if that improves readability. Code example 8, indents line 3 so that the comparisons match up.

```java
1  if (commandOption == 'f' || commandOption == 'c' || commandOption == 'd' || commandOption == 'g')
    //WRONG: Because the text is wrapped.
2  if (commandOption == 'f' || commandOption == 'c' ||
3     commandOption == 'd' || commandOption == 'g')
    //Correct: Two physical lines, one logical line.
```
Fixing Too Long a Line Example 4

If a string literal is too long to fit on a single line then it should be broken, not wrapped, but left as a single logical statement:

```java
String prompt = "Whose woods these are I think I know,
his house is in the village though.";
//WRONG: Because the text is wrapped.

String prompt = "Whose woods these are I think I ";
prompt += "know, his house is in the village though.";
//WRONG: Because one literal is broken into two logical lines.

String prompt = "Whose woods these are I think I 
+ "know, his house is in the village though.";
//Correct: Two physical lines, one logical line.
```

Quiz: Coding Standard

Which line does NOT follow the standard?

a: `for (i=0; i<10; i++)`
b: `{ int c = i*10;`
c: `if (c == 30)`
d: `c=c+6;`
e: `else if (c == 40) c = c-6;`
}`
import javax.swing.JFrame;

public class MyClass
{
   private static int myClassIntVariable1;
   private static int myClassIntVariable2;

   public MyClass()
   { //Executable Code
   }

   public int myMethod1(int x, int y)
   { //Executable Code
   }

   public static void main(String[] args)
   { //Executable Code
   }
}

---

**Principle of Least Privilege**

In computer science, the *Principle of Least Privilege* requires that in a particular abstraction layer of a computing environment, every module must be able to access only such information and resources that are necessary for its legitimate purpose.

*Variables* used in only one method shall be *local variables*.

*Variables* used in more than one method within a class shall be *private class variables*.

*A data class* (a class with no methods) is the only type of class that may have public variables.
Write Self-Documenting Code

- **Self-documenting code** uses well-chosen names and has a clear style.

- The program's purpose and workings should be obvious to any programmer who reads the program, *even if the program has no comments*.

- To the extent that is possible, strive to make your programs self-documenting.

Clean Code *As You Type*

- Use clean coding standards *as you author the code*.

- Eclipse will let you enter sloppy code. Then, with a click or two: 🙁 you have neatly formatted code. *In this path lies ruin!*

- To write code of nontrivial length, your written code, as seen in the editor, must become an extension of your mind.

- If your code is a mess, your thoughts will be a mess.
Setting Code Style in Eclipse

Window → Preferences

Note: "Preferences" can also be selected under the File menu and under the Project menu - Those are different preferences.

1. Open Preferences
2. Select Formatter
3. Select Indentation

By using *spaces only*, code still looks good:
- When seen in other Java environments.
- When copy/pasted into e-mail.
Code Style → Formatter → Indentation

The upper part of the Indentation tab of the Code Style Formatter dialog is shown on previous slide.

Code Style: Braces

Eclipse
→ Window
→ Preferences
→ Java
→ Code Style
→ Formatter
→ Braces
Code Style: Control Statements

- Insert new line before 'else' in an 'if' statement
- Insert new line before 'catch' in a 'try' statement
- Insert new line before 'finally' in a 'try' statement
- Insert new line before 'while' in a 'do' statement

'if else'
- Keep 'then' statement on same line
- Keep 'else' statement on same line
- Keep 'else if' on one line
- Keep 'return' or 'throw' clause on one line

Example:
```java
if (true)
    return;
else if (false)
    return;
else
    return;
}
```

void foo(int state)
{
    if (true) return;
    if (true) return;
    else if (false) return;
    else return;
}

Code Style Line Wrapping

- Keeps code looking good:
  - When seen in other Java environments.
  - When copy/pasted into e-mail.
Applying Code Style

1. Set Active profile to CS-152.
2. Select source code.
3. Right Click in source code, and Select: Source → Format.

As You type: Automatically Close...

Choose whatever works best for you.

Sometimes picks the wrong import.
Quiz: Coding Standard

Which line does NOT follow the standard?

```java
for (i=0; i<10; i++)
{
    char c = inStr[i];
    if (c == '+') c=a+b;
    else if (c == '*') c = a*b;
    else if (c>='0' && c<='9')
    { for (j=0; j<c; j++)
        { System.out.print("j="+ j);
        }
        System.out.print("\n");
    }
}
```

Quiz: Coding Standard

Which lettered line does NOT follow the code standard?

a) int n = 10;
b) for (i=2; i<n; i++)
{
c) if (n % i == 0)
{
d) System.out.print(i + " divides " + n);
}
e) else
{
    System.out.print("bad " + i);
}
```
Clean Up Your Code

_Leave No Warnings:_

```java
import java.awt.Container;
import java.awt.Dimension;

The import java.awt.Dimension is never used

public static void main(String[] args) {
    GUI_Frame guiAnt = new GUI_Frame("Ant",
    GUI_Frame guiBat = new GUI_Frame("Bat",
    GUI_Frame guiCat = new GUI_Frame("Cat",
```

The local variable guiAnt is never read

Quiz: Coding Standard

If the non-lettered lines do follow the code standard, then which _lettered_ line does NOT follow the standard?

a) `System.out.println("Lets look at r.");`
   `if (r < 100)`
   `{`  
   b) `System.out.println(r + " is a ");`
       `System.out.println("dark red value.");`
   `}`
   c) `else`
   d) `{ System.out.println(r + " is a ");`
   e) `System.out.println("bright red.");`
   `}`
Error / Warning Reporting: Code Style

- Non-static access to static member:
- Indirect access to static member:
- Unqualified access to instance field:
- Access to a non-accessible member of an enclosing type:
- Parameter assignment:
- Non-externalized strings (missing/unused $NON-NLS$ tag):
- Undocumented empty block:
- Resource not managed via try-with-resource (1.7 or higher):
- Method with a constructor name:
- Method can be static:
- Method can potentially be static:

Error / Warning: Potential Problems

- Comparing identical values ('x = = x'):
- Assignment has no effect (e.g., 'x = x'):
- Possible accidental boolean assignment (e.g., 'if (a = b)'):
- Using a char array in string concatenation:
- Inexact type match for vararg arguments:
- Empty statement:
- Unused object allocation:
- Incomplete 'switch' cases on enum:
- 'Switch' is missing 'default' case:
- 'Switch' case fall-through:
- Hidden catch block:
- 'finally' does not complete normally:
- Dead code (e.g., 'if (false)'):
- Resource leak:
- Potential resource leak:
- Serializable class without serialVersionUID: