CS 351
Design of Large Programs
Code Standards

Instructor:
Joel Castellanos

e-mail: joel@unm.edu

Web: http://cs.unm.edu/~joel/

Office: Farris Engineering Center (FEC) room 319

HOW STANDARDS PROLIFERATE:
(SEE: A/C CHARGERS, CHARACTER ENCODINGS, INSTANT MESSAGING, ETC)
SITUATION: THERE ARE 14 COMPETING STANDARDS.
14?! RIDICULOUS! WE NEED TO DEVELOP ONE UNIVERSAL STANDARD THAT COVERS EVERYONE'S USE CASES.
YEAH!
SOON:
SITUATION: THERE ARE 15 COMPETING STANDARDS.

1/17/2017

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.

2

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:.....

3

CS-351 Code Standard

- All CS-351 assignments must follow the great and hallowed CS-351 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.

4

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.

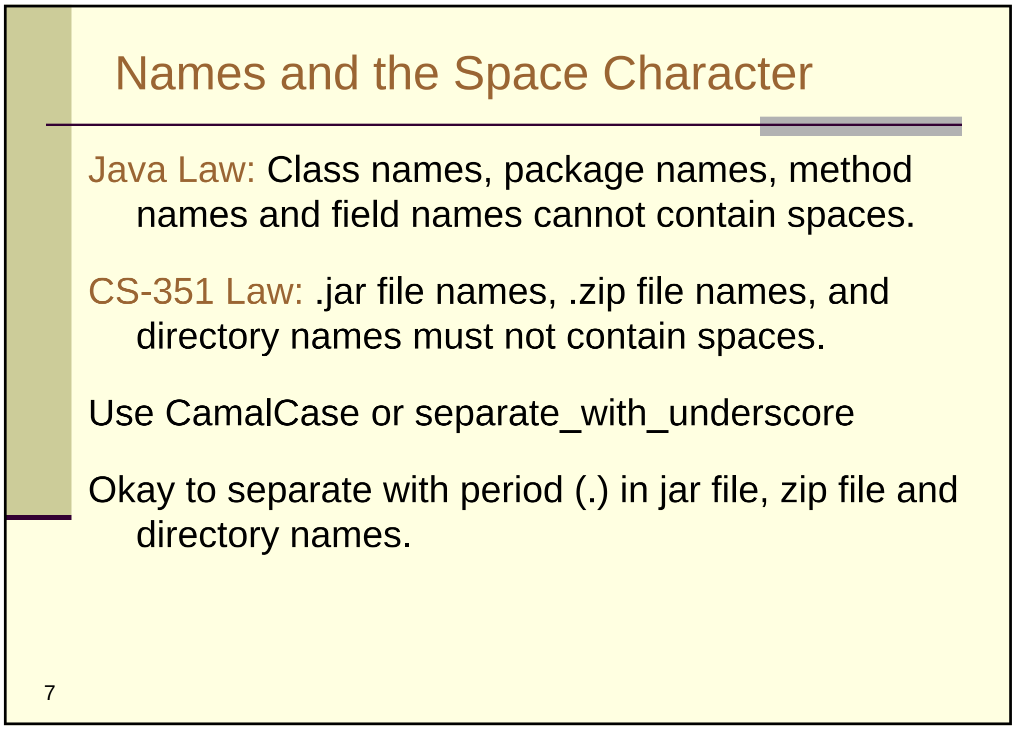
   Often when each student is allowed to define his or her own standard, students switch standards multiple times in a single project. It is tedious for a grader to deduce each person's standard and then check for self-consistency.
3. Learning to adhere consistently to a coding standard is a good practice.

5

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 l.
- All methods will be given descriptive names.
- All class names shall begin with an uppercase letter.

6



Names and the Space Character

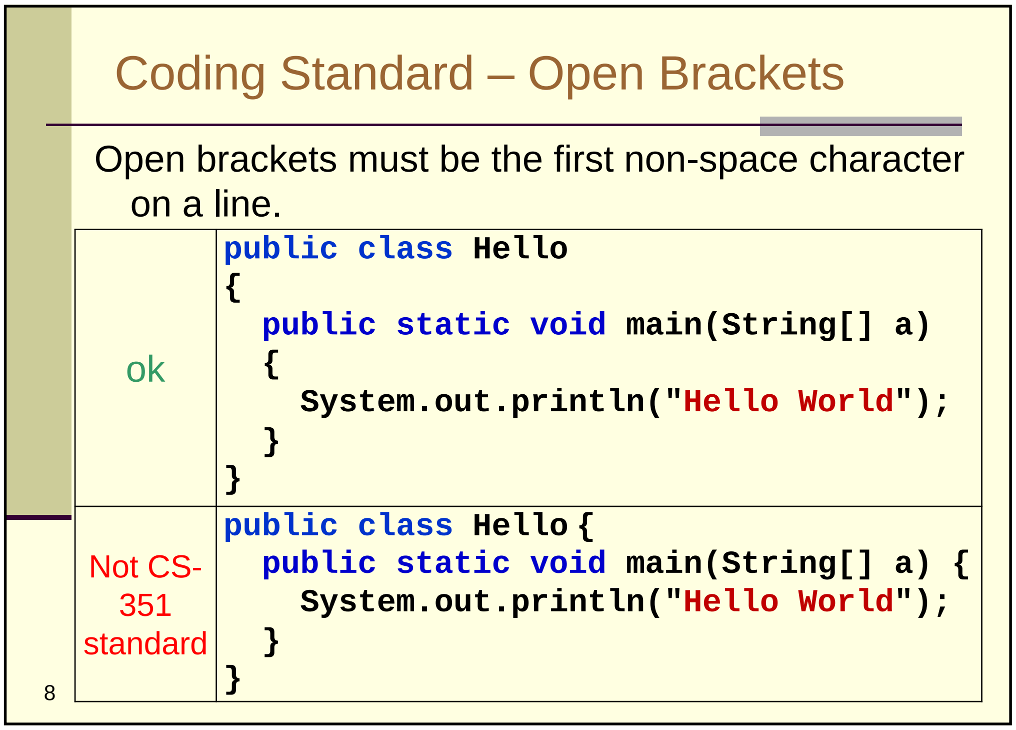
Java Law: Class names, package names, method names and field names cannot contain spaces.

CS-351 Law: .jar file names, .zip file names, and directory names must not contain spaces.

Use CamalCase or separate_with_underscore

Okay to separate with period (.) in jar file, zip file and directory names.

7



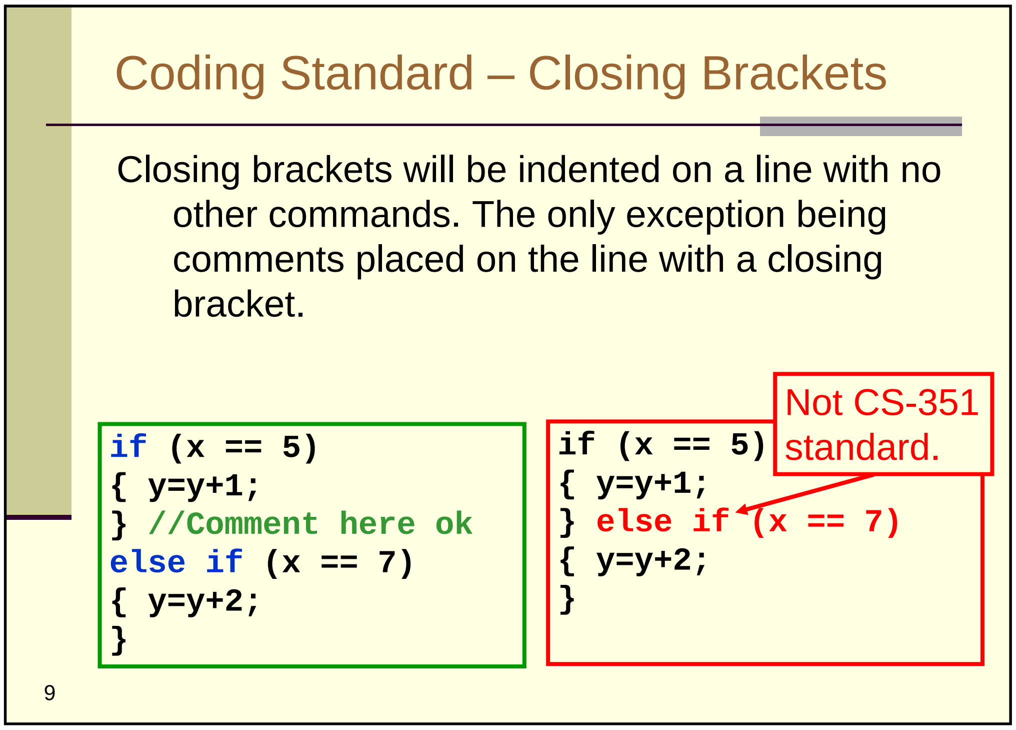
Coding Standard – Open Brackets

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

ok	`public class Hello`
	`{`
	`  public static void main(String[] a)`
	`  {`
	`    System.out.println("Hello World");`
	`  }`
	`}`

Not CS-351 standard	`public class Hello {`
	`  public static void main(String[] a) {`
	`    System.out.println("Hello World");`
	`  }`
	`}`

8



Coding Standard – Closing Brackets

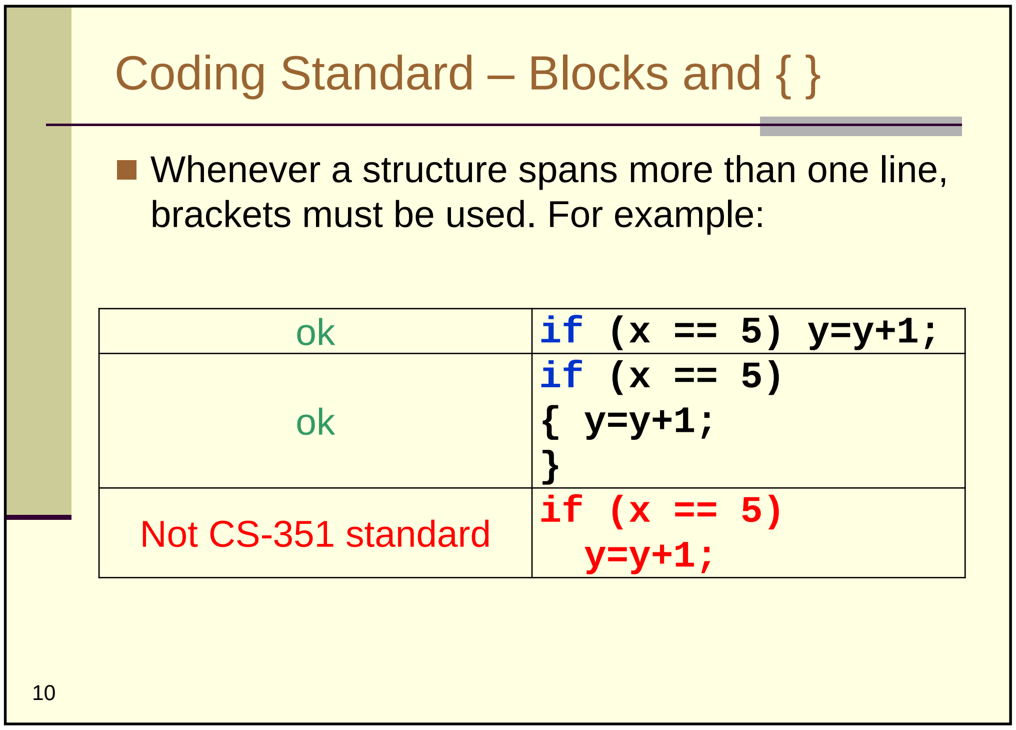
Closing brackets will be indented on a line with no other commands. The only exception being comments placed on the line with a closing bracket.

```
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;
}
```

Not CS-351 standard.

9



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-351 standard	`if (x == 5)`
	`  y=y+1;`

10

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.

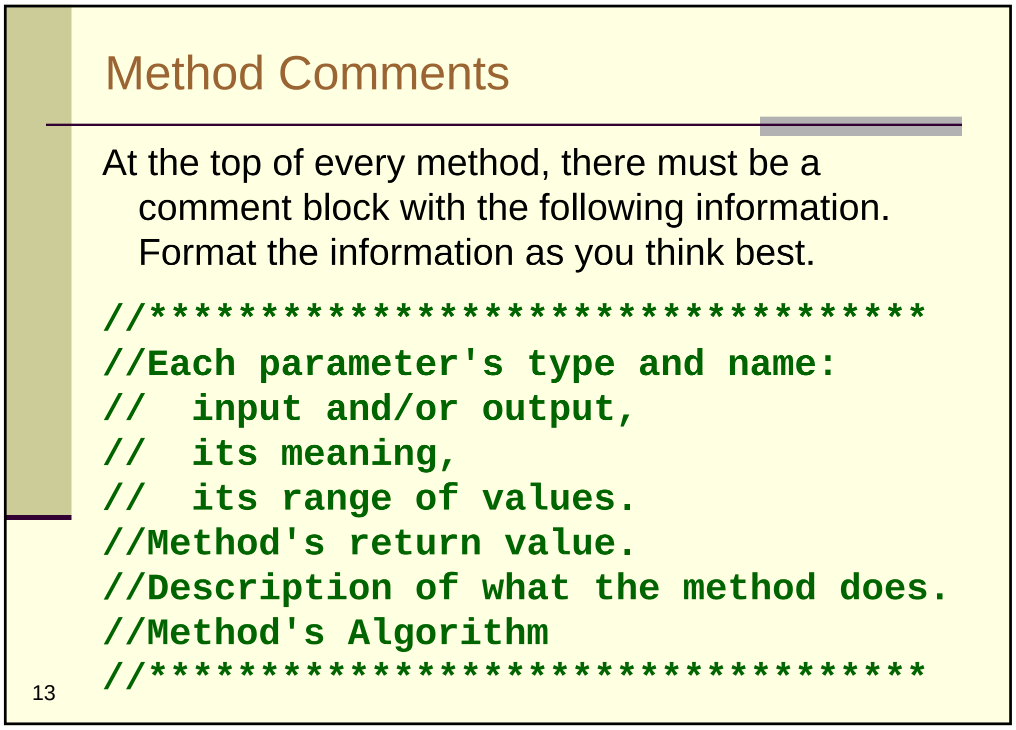
11

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.
//***********************************
```

12

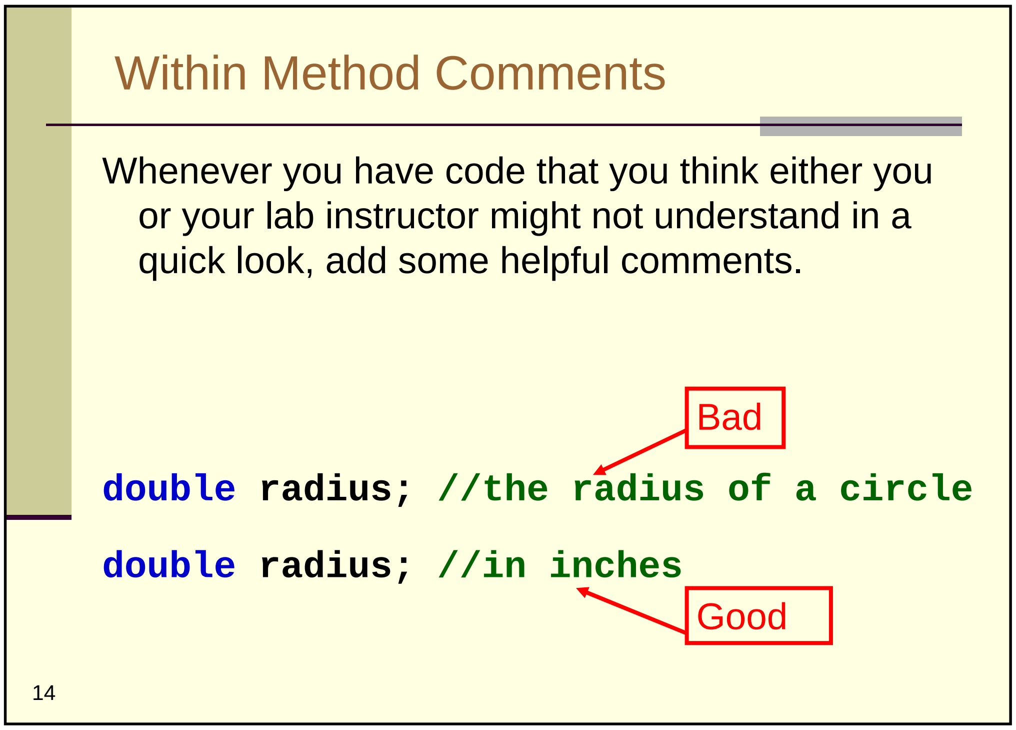


Method Comments

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
//*************************************
```

13



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.

Bad

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

double radius; //in inches
```

Good

14

380 Character Line Max

No line shall be more than 380 characters.

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

```
if (getVolume(length1, width1, height1) >
getVolume(length2, width2, height2)) System.out.println
("box 1 is bigger"); else System.out.println ("box 2 is
bigger");
```

```
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");
}
```

Stored in register

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 **`stack[topOfStack]`** or **`stack[topOfStack-2]`**, then Line 2 and 3 require only a single change while line 1 requires 4 changes.

```
1 if (stack[topOfStack - 1] == '*' || stack[topOfStack
  - 1] == '+' || stack[topOfStack - 1] == '-' ||
  stack[topOfStack - 1] == '/')
```

`//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 in to 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 it that improves readability. Code example 8, indents line 3 so that the comparisons match up.

```
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.
```

17

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:

```
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.
```

18

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;
   }
```

19

Class Layout

```
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
 }
}
```

import list

All class variables: public then private.

Constructor(s)

Methods(s): public then private

Methods: main

20

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.

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

Fields used in more than one method within a class shall be **`private`** ***class variables***. An exception is a field that is declared **`final`** may be **`public`** or **`protected`**.

A data class (a class with no methods) is the only type of class that may have **`public`** fields.

21

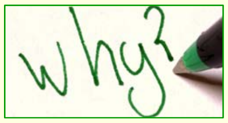
Protected Access Level

- In Java, variables declared **`protected`** are visible outside the class but not outside the package except within subclasse
- The use of **`protected`** should follow similar guidelines as the use of **`public`**. That is, only used it for fields that are constants or data classes and minimize the number of methods that use it.

22

Getters and Setters

- In Java, when outside access is needed for a field, the class should provide a getter and/or setter for the field.
- For example, **`JFrame.getContentPane()`**, **`JFrame.getJMenuBar()`**, **`JFrame.setJMenuBar()`**, …
- DO NOT auto generate a getter and setter for every field in your class.



- Only create a getter or setter when there is actually a use for that getter or setter.

Package Names

- At the level of CS-351, all your code should be in named packages. However, do not use long chains of sparsely populated directories:

```
usa.edu.unm.cs.351.2015.fall.zombieHo
use.theNutters.gui.mvc.nw.rsc.qcm.the
NuttersSlider.java
```

- Do not, for example, make a separate .gui package unless it has at least 4 classes.
- Do not use **`com.company`**

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.

25

Clean up Debug Output

- When your program is run, it is okay if a few lines of status/debug info is displayed. For example, if the user resizes the window, you might want to have your program print the new inside window size.
- However, your program should not be printing pages of debug.
- Either comment out your debug statements or protect them with: if (DEBUG_GUI) or some equivalent. Be sure to turn in a version with all the debug flags set to false.

26

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: POOF! 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.

27

Quiz: Coding Standard

Which line does NOT follow the standard?

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

28

Quiz: Coding Standard

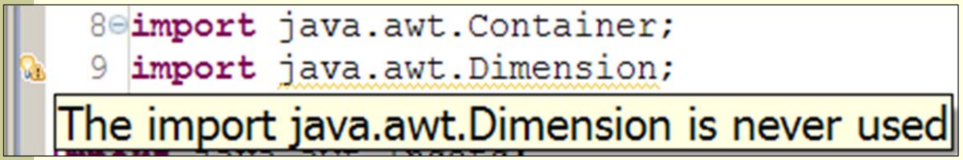
Which ***lettered*** line does NOT follow the code standard?

```
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);
   }
 }
```

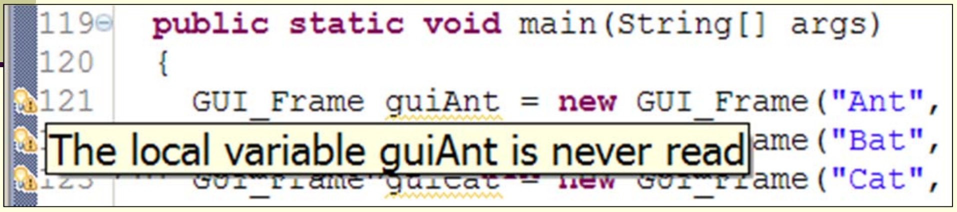
Clean Up Your Code

Leave No Warnings:



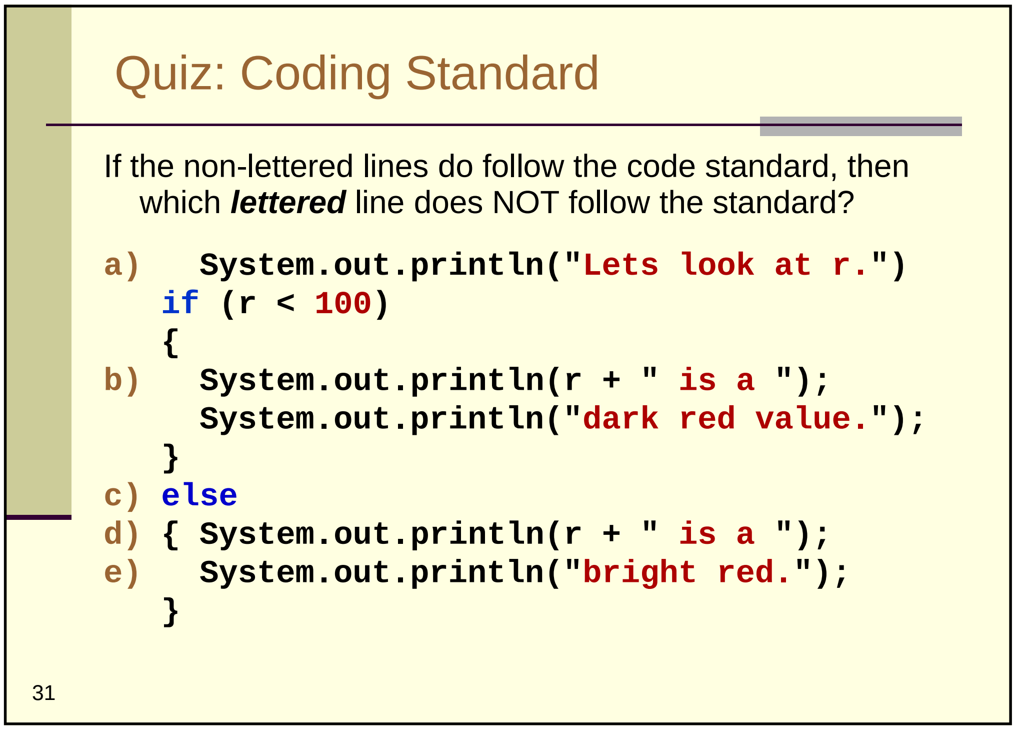
```
import java.awt.Container;
import java.awt.Dimension;
```

The import java.awt.Dimension is never used



```
public static void main(String[] args)
{
  GUI_Frame guiAnt = new GUI_Frame("Ant",
                           ame("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.");
   }
```

31