Code Layout, Inheritance & Interfaces

The goal of code layout/formatting is to show logical structure.

Good layout is shows intention, is consistent, improves readability, and withstands modification.
Which is best?

A)    for(int i=0;i<str.length();i++){
B)    for (int i=0; i<str.length(); i++) {
C)    for (int i = 0; i < str.length(); i++) {
D)    for (int i = 0 ; i < str.length() ; i ++) {
E)    for( int i = 0 ; i < str.length() ; i ++ ){
Which is better?

A) for (int i = 0; i < args.length; i++)
B) for (int i = 0; i < args.length; i++)
C) Both are fine
D) Both are lacking
Which is better?

A) if (game[i][index] != c)
B) if ( game[ i ][ index ] != c )
C) Both are fine
D) Both are lacking
Which is better?

A) char [][] game = new char[3][3];
B) char [][] game = new char[3][3];
C) Both are fine
D) Both are lacking
Hmmm…

- I like spacing operands like the following:
  \[
  \text{int } x = a + b + c + d + 17; \\
  \]

- But in the below, I personally prefer the second option:
  \[
  \text{data}[i][i] = \text{data}[i-1][i-1]; \\
  \text{data}[i][i] = \text{data}[i-1][i-1]; \\
  \]

  \[
  \text{int index} = a + b + \ldots \\
  \text{data}[\text{index}] \\
  \]
Are you familiar with the ternary operator?

```java
if (a) {
    x = b;
} else {
    x = c;
}
```
What is wrong with this?

```c
int foo = (a + b == 10) ? c : (d + e);
```
Which is better?

A) \text{int parenthesis;}
\text{parenthesis} = 0;

B) \text{int parenthesis} = 0;

C) Both are fine

D) Both are lacking
\text{int parenthesis = 0;

for ( \ldots ) \exists
  \text{int parenthesis =}
  3
  \ldots \text{parenthesis.}
Which is better?

A) `int paren = 0, eqnLength = eqn.length();`

B) `int paren = 0;
    int eqnLength = eqn.length();`

C) Both are fine

D) Both are lacking
What is wrong with this?

aFunction(j, k); j++; k++;
Which is better?

A)  
```java
if (three) {
    System.out.println("Valid: "+ value);
}
```

B)  
```java
if (three) {
    System.out.println("Valid: "+ value);
} else {
    System.out.println("Invalid");
}
```

C) Both are fine  
D) Both are lacking
Which is better?

A)
```java
if (three) {
    System.out.println("Valid");
} else {
    System.out.println("Invalid");
}
```

B)
```java
if (three)
    System.out.println("Valid");
else
    i++;  // Missing increment
    System.out.println("Invalid");
```

C) Both are fine
D) Both are lacking
Which is best?

A) if (prev_type==type&&type!=1&&type!=2) {
B) if (prev_type == type && type != 1 && type != 2) {
C) if ((prev_type == type) && (type != 1) && (type != 2)) {

D) All are fine
E) All are lacking
Inheritance

- Super-type / Sub-type (extends in Java)
  - IsA relationship; the sub-type isA version of super-type

- Abstract:
  - Cannot be instantiated, but describes the interface of what a given type can do.

- Protected:
  - Public to my sub-classes (transitively), private to others
Casting in Java

- What if you have an object in a super type and you want to access its sub-type only functionality?

- If you _know_ what the sub-type is, just cast it:
  - SuperType x = new SubType();
  - SubType xAsSubType = (SubType)x;  // will except if wrong

- If you aren’t sure, then ask: instanceof
  - if (x instanceof SubType) {
  - then cast

```java
char c;
""" + c
"" + c.toString()
```
How does it work?

- See scribble along with handout / watch the video
Interfaces vs. Abstract Base Classes

- Java objects can only extend one other class
  - “single inheritance”
- Sometimes logical inheritance hierarchies aren’t trees

- Java provides Interfaces
  - You can ‘implement’ any number of interfaces
  - List and Map are interfaces, while ArrayList and HashMap are classes

```java
List<String> list = new ArrayList<>();
```