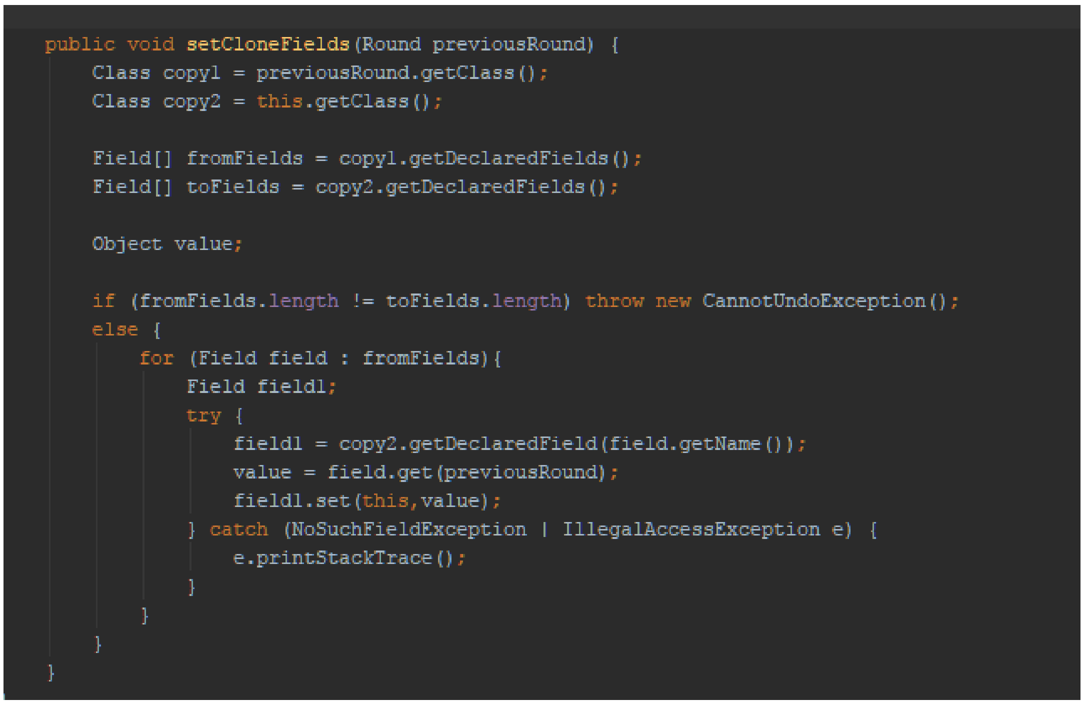
Smart Code highlight:

I think given I started undo at about 10pm on Sunday night I thought my implementation was quite good. I used a stack that contained Rounds. These rounds were pushed onto the stack every successful move. If undo was called, it would pop off the round on the top of the stack, I.E the previous state of the round. Simple enough.
I needed a way to reassign the fields of my current round instance to the fields of the one I had just popped off. I immediately thought of using Java reflection. Hence, the following code:



```java
public void setCloneFields(Round previousRound) {
    Class copy1 = previousRound.getClass();
    Class copy2 = this.getClass();

    Field[] fromFields = copy1.getDeclaredFields();
    Field[] toFields = copy2.getDeclaredFields();

    Object value;

    if (fromFields.length != toFields.length) throw new CannotUndoException();
    else {
        for (Field field : fromFields){
            Field field1;
            try {
                field1 = copy2.getDeclaredField(field.getName());
                value = field.get(previousRound);
                field1.set(this,value);
            } catch (NoSuchFieldException | IllegalAccessException e) {
                e.printStackTrace();
            }
        }
    }
}
```

I thought in the grand scheme of things this was good because I was using something that beyond the SWEN221 labs I hadn't really seen the need for and two, it meant any change to the round class meant that this code would still work and wouldn't break the undo functionality.

I suppose I could have commented on the fact I parsed in all my data for my pieces so I didn't need to create heaps of very similar code. Or how I used a recursive descent parser to return a generic ParseNode to be utilised in a command pattern.