

Improve RentalAgreement Design

I have a Java class in my project that I want to follow Java Solid Principles. Infer the class's responsibility and look for ways to improve it, ONLY if applicable. Ignore other context.

```
```package com.toolstore.demo.model;
```

```
import com.fasterxml.jackson.annotation.JsonFormat;
import lombok.Builder;
import lombok.Getter;
```

```
import java.math.BigDecimal;
import java.time.LocalDate;
import java.time.format.DateTimeFormatter;
import java.text.NumberFormat;
import java.util.Locale;
```

```
@Getter
@Builder
public class RentalAgreement {
```

```
 private String toolCode;
 private String toolType;
 private String toolBrand;
 private int rentalDays;
```

```
 @JsonFormat(pattern = "MM/dd/yy")
 private LocalDate checkoutDate;
```

```
 @JsonFormat(pattern = "MM/dd/yy")
 private LocalDate dueDate;
```

```
 private BigDecimal dailyRentalCharge;
 private int chargeDays;
 private BigDecimal preDiscountCharge;
 private int discountPercent;
 private BigDecimal discountAmount;
 private BigDecimal finalCharge;
```

```
}```
```

## Inferred responsibility

RentalAgreement appears to be a **domain model / DTO** that represents the output of a rental checkout: tool identity + rental period + computed charges/discounts. As written, it's essentially an immutable data holder (via `@Getter` + `@Builder`), which is a good, single responsibility.

## SOLID review (only improvements if applicable)

### 1) Single Responsibility Principle (SRP)

- **Currently OK:** There's no behavior; it just holds agreement data.
- **One small concern:** `@JsonFormat` ties the domain object to a specific serialization format (Jackson + date pattern). That's not "wrong", but it does couple the model to an infrastructure concern. If you want stricter SRP/clean architecture, consider moving formatting concerns to a DTO or API layer model.

If you have separate API models:

- Keep RentalAgreement pure (no Jackson annotations).
- Create RentalAgreementResponse (or similar) with `@JsonFormat`.

If you don't have that layering, leaving `@JsonFormat` is pragmatic and not a SOLID violation in many codebases.

### 2) Interface Segregation / Dependency Inversion

- Not really applicable here (no interfaces/dependencies).

### 3) Open/Closed / Liskov

- Not really applicable (no inheritance/polymorphism).

## Concrete improvements (only if you want them)

### A) Remove unused imports (definitely applicable)

You import `DateTimeFormatter`, `NumberFormat`, `Locale` but don't use them. Clean these up to reduce noise and avoid implying formatting responsibility.

## **B) Consider stronger types for money/percent (optional)**

If this class is used widely in the domain, you can reduce invalid states:

- Replace int discountPercent with a tiny value object (e.g., DiscountPercent that enforces 0..100), or at least validate at construction time elsewhere.
- Money is already BigDecimal (good), but ensure you have a consistent scale/rounding policy—typically enforced in the pricing/charge calculation service, not here.

## **C) Consider whether tool identity should be a separate value object (optional)**

If toolCode/toolType/toolBrand always travel together, you could model:

- ToolInfo { code, type, brand }  
and embed it. This can improve cohesion, but only if it matches your domain usage.

## **Bottom line**

As shown, the class is already largely SRP-compliant as a data model. The only clearly applicable improvement is **removing unused imports**. The only potentially SOLID-relevant design tweak is **moving @JsonFormat out** if you want a strict separation between domain and serialization concerns.

---