Das Creational Pattern **Builder** anhand des Beispiels eines Cat-Builders

Steckbrief:

- **Name:** Builder (deutsch: Erbauer)
- **Art:** Creational Pattern (deutsch: Erzeugungsmuster)
- **Zweck:** Komplexe Objekte Schritt für Schritt konstruieren
- **Prinzip:** Objekt-Konstruktion vom Objekt selbst trennen

Vorteile:

- Lesbarer Code durch schrittweise Objektkonstruktion
- Konsistenter Buildcode unabhängig von interner Repräsentation
- Neue Repräsentationen lassen sich einfach durch neue Builderklassen einbauen

Nachteile:

- Kann Code-Komplexität bei zusätzlichen Klassen erhöhen
- Boilerplate-Code möglich

Einsatzgebiete:

- Objekte mit vielen Parametern
- Konstruktion erfordert mehrere Schritte
- Verschiedene Objektvarianten aus gleichem Prozess

Warum existiert dieses Pattern?

Das Builder Pattern existiert, um…

- …komplexe Objekte übersichtlich und schrittweise zu erstellen
- …ohne viele unübersichtliche Konstruktoren
- …ohne Risiko von inkonsistenten Objekten

Ein Builder macht die Objekterstellung explizit, flexibel und wartbar.

Code-Auszüge anhand eines Cat-Builders:

```
11      myLittleCatTroop.add(new Cat.CatBuilder().
12              setName(name:"Tommy").
13              setAge(age:9).
14              setCatGender(CatGender.MALE).
15              setPersonality(CatPersonality.CUDLY).
16              setRace(CatRace.SIAM).
17              setFurColor(CatFurColor.GREY).
18              setFurPattern(CatFurPattern.SOLID).
19              setEyeColor(CatEyeColor.BLUE).
20              setFavouriteToy(CatToy.LASER_POINTER).
21              setFavouriteConsumable(CatConsumable.SCRAMBLED_EGGS).
22              build());
```

```
115     private Cat(){
116         name = DEFAULT_NAME;
117         age = DEFAULT_AGE;
118         catGender = DEFAULT_CAT_GENDER;
119         personality = DEFAULT_PEERSONALITY;
120         race = DEFAULT_RACE;
121         furColor = DEFAULT_FUR_COLOR;
122         furPattern = DEFAULT_FUR_PATTERN;
123         eyeColor = DEFAULT_EYE_COLOR;
124         favouriteToy = DEFAULT_FAVOURITE_TOY;
125         favouriteConsumable = DEFAULT_FAVOURITE_CONSUMABLE;
126     }
```

```
152     public CatPersonality getPersonality() {
153         return personality;
154     }
155
156     private void setPersonality(CatPersonality personality) {
157         this.personality = personality;
158     }
```

```
83      public CatBuilder setFavouriteConsumable(CatConsumable favouriteConsumable){
84          cat.setFavouriteConsumable(favouriteConsumable);
85          return this;
86      }
87
88      public Cat build(){
89          return cat;
90      }
```

```
1   package CatEnums;
2
3   public enum CatPersonality {
4       FRIENDLY(catPersonality:"friendly"),
5       ANGRY(catPersonality:"angry"),
6       SHY(catPersonality:"shy"),
7       CUDLY(catPersonality:"cudly"),
8       SLEEPY(catPersonality:"sleepy"),
9       HUNGRY(catPersonality:"hungry"),
10      NOSY(catPersonality:"nosy"),
11      MOODY(catPersonality:"moody");
12
13      private final String catPersonality;
14
15      CatPersonality(String catPersonality){
16          this.catPersonality = catPersonalit
17      }
18
19      @Override
20      public String toString() {
21          return catPersonality;
22      }
23  }
```

Gesamter Code unter diesem Link auf GitHub.