THE BUILDER PATTERN

THE INTENTION OF THE BUILDER PATTERN IS TO FIND A SOLUTION TO THE TELESCOPING CONSTRUCTOR ANTI-PATTERN

```
class Reading {

  private final int id;
  private final int sourceId;
  private final String name;
  private final Date time;
  private final double value;

  public Reading(int id, int sourceId, String name,
                 Date time, double value) {
    this.id = id;
    this.sourceId = sourceId;
    this.name = name;
    this.time = time;
    this.value = value;
  }
}
```

```
class Reading {
  /* ... */
  public Reading(int id, int sourceId, String name,
                 Date time) {
    this(id, sourceId, name, time, 0.0);
  }
}
```

```
class Reading {

  /* ... */

  public Reading(int id, int sourceId, String name){
    this(id, sourceId, name, new Date());
  }
}
```

```
class Reading {
  /* ... */
  public Reading(int id, int sourceId){
    this(id, sourceId, “UNNAMED”);
  }
}
```

```
class Reading {

  /* ... */

  public Reading(int id){
    this(id, -1);
  }
}
```

```
class Reading {

  /* ... */

  public Reading(){
    this(-1);
  }
}
```

```
public Reading(){}
public Reading(int){}
public Reading(int, int){}
public Reading(int, int, String){}
public Reading(int, int, String, Date){}
public Reading(int, int, String, Date, double){}
```

CODE REVIEW

INCREASED BOILERPLATE

HIGHLY COUPLED CONSTRUCTORS

DECENTRALISED DEFAULTS

BUT...

NOW WHAT HAPPENS WHEN YOU ONLY WANT TO SUPPLY A DATE?

```
class Reading {
  /* ... */
  public Reading(Date time){
    this(-1, -1, “UNNAMED”, time);
  }
}
```

BUT...

NOW WHAT HAPPENS WHEN YOU ONLY WANT TO SUPPLY A SOURCE ID?

```
class Reading {

  /* ... */

  public Reading(int sourceId){
    this(-1, sourceId);
  }
}
```

```
class Reading {

  /* ... */

  public Reading(int sourceId){
    this(-1, sourceId);
  }
}
```

THE BUILDER PATTERN
DEMONSTRATION

OTHER BENEFITS

IMPROVED READABILITY

COMPOSITION OF COMPLEX OBJECTS

OPTIMISE OBJECT CREATION

```
private void buildOrder(Customer customer) {
  Order o1 = new Order();
  customer.addOrder(o1);

  OrderLine line1 = new OrderLine(6, Product.find("TAL"));
  o1.addLine(line1);

  OrderLine line2 = new OrderLine(5, Product.find("HPK"));
  line2.setSkippable(true);
  o1.addLine(line2);

  OrderLine line3 = new OrderLine(3, Product.find("LGV"));
  o1.addLine(line3);

  o1.setRush(true);
}
```

```
private void buildOrder(Customer customer) {
   customer.newOrder()
      .with(6, "TAL")
      .with(5, "HPK").skippable()
      .with(3, "LGV")
      .priorityRush();
}
```

CONSTRUCTION OF REPRESENTATIONS & COMMUNICATION ACROSS DOMAINS

```
select(“*”)
  .from(“users”)
  .where(“username”, eq(“james”))
  .and(“isAdmin”, eq(true))
  .limit(1);
```

```
SELECT * FROM users
WHERE username = ‘james’
AND isAdmin = true
LIMIT 1
```

```
Json.obj(
  "users" -> Json.arr(
    Json.obj(
      "name"    -> "james",
      "isAdmin" -> true
    )
  )
)
```

```
{
  “users”: [
    {“name”: “james”, “isAdmin”: true}
  ]
}
```

CAVEAT EMPTOR

BUILDER CODE CAN BE COMPLEX

INCOMPLETE OBJECT CONSTRUCTION

THE BUILDER PATTERN