factor10

Just start (with Value Objects)

Katharina Damschen

Coding Architect and Consultant

Wal_172619 via pixabay

factor10

Marvin Meyer via unsplash

?

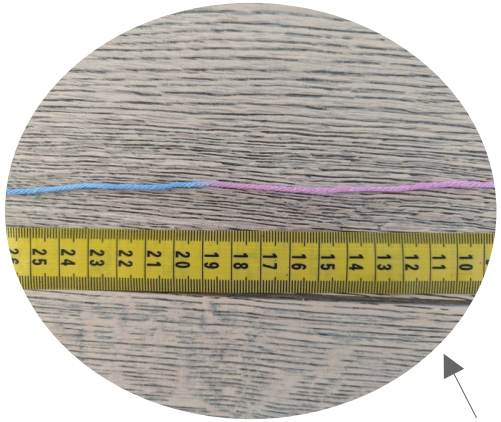
Pop&Zebra via unsplash and Alexa via pixabay

TK McLean via pixabay

Knitting Project

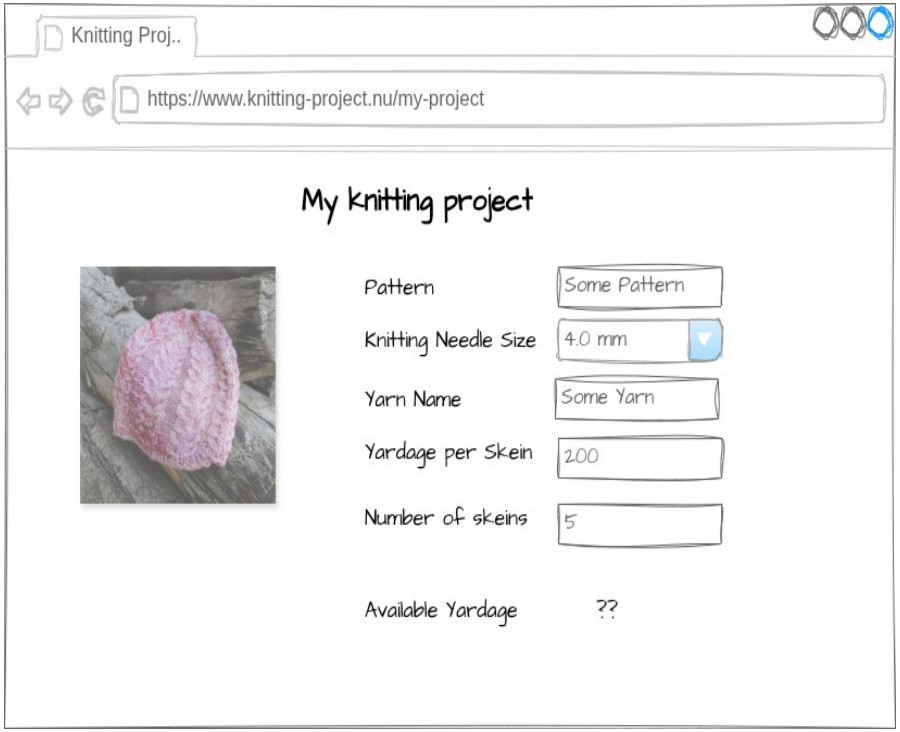
Yarn

Skein



Yardage

The task



Suggested yarn

To knit this top, you'll need fingering weight yarn (I recommend a yarn that doesn't stretch too much.

For size A (B, C, D, E) F (G, H, I, J) you'll need 490 (530, 600, 670, 730) 800 (870, 950, 1120, 1200) meters of yarn.

Please, have in mind that the yarn needed may vary depending on how long you want your top.

The 'solution'?

factor10

```
object YarnCalculator {
    fun calculateTotalYardage(yardagePerSkein: Long, skeinCount: Int) =
        yardagePerSkein * skeinCount
}
```

```
@Test
fun `calculate total`() {
    val total = YarnCalculator.calculateTotalYardage(200, 5)

    assertEquals(1000, total)
}
```

factor10

Event Storming?

Ubiquitous Language?

Domain-driven Design

Domain Model?

Context Map?

Tactical Patterns?

Entities

Objects that are defined primarily by their identity (not by their attributes).

Entities

```
class KnittingProject(
    val id: UUID = UUID.randomUUID(),
    var yardagePerSkein: Long,
    var skeinCount: Int
) {
    override fun equals(other: Any?): Boolean {
        if (this === other) return true
        if (other !is KnittingProject) return false
        return id == other.id
    }

    override fun hashCode(): Int = id.hashCode()
}

// val total = project.yardagePerSkein * project.skeinCount
```

Anemic Models

- Domain objects are just data containers.
- Services contain all business rules.
- No encapsulation or invariant protection.

- Everything modeled as an Entity.

Value Objects

Objects without a conceptual identity that are primarily defined by their attributes.

Value Objects

Immutable object =
object whose state
cannot be changed after
its creation.

Value Objects

```
data class Skein(val yardage: Long) {
    init {
        require(yardage >= 0) { "Yardage must be positive" }
    }
}
```

No general rules

An object that is an entity in one context is not neccessarily an enitity in another context.

```
class KnittingProject private constructor(
    val id: UUID,
    val skeins: List<Skein>
) {
    constructor(id: UUID = UUID.randomUUID()) : this(id, emptyList())

    fun addSkeins(skein: Skein, amount: Int): KnittingProject {
        require(amount > 0) { "Amount must be positive" }

        val newSkeins = List(amount) { skein }
        return KnittingProject(id, skeins + newSkeins)
    }

    fun calculateTotalAvailableYardage(): Long =
        skeins.sumOf { it.yardage }

    // equals & hashCode unchanged
}
```

Value Objects everywhere

factor10

No simple types in the domain model.

Replace with small Value Objects that validate their invariants in the constructor.

```
data class Skein(val yardage: Long) {
    init {
        require(yardage >= 0) { "Yardage must be positive" }
    }
}
```

```
data class Skein(val yardage: Yardage)
```

```
value class Yardage(val value: Long) {
    init {
        require(value >= 0) { "Yardage must be positive." }
    }
}
```

```
value class Yardage(val value: Long) {
    init {
        val result = validate(value)
        require(result is ValidationResult.Success) { (result as ValidationResult.Failure).error }
    }

    companion object {
        private const val MIN_YARDAGE = 0L

        fun validate(value: Long): ValidationResult {
            return when {
                value < 0 ->
                    ValidationResult.Failure("Yardage must be positive.")

                else -> ValidationResult.Success
            }
        }
    }
}
```

```
// use a result wrapper to be able to return
// a specific message, not just true or false
sealed interface ValidationResult {
    object Success : ValidationResult
    data class Failure(val error: String) : ValidationResult
}
```

Discovering hidden requirements

factor10

```
companion object {
    private const val MIN_YARDAGE = 0L
    private const val MAX_YARDAGE = 50000L

    fun validate(value: Long): ValidationResult {
        return when {
            value < MIN_YARDAGE ->
                ValidationResult.Failure("Yardage must be positive.")

            value > MAX_YARDAGE ->
                ValidationResult.Failure("Yardage must not exceed $MAX_YARDAGE.")

            else -> ValidationResult.Success
        }
    }
```

Yarn | Details | Colorways | Buying Options (6) | Stashes (43) | Pa

by knittingfever-ish

Rambouillet DK

from Juniper Moon Farm

Juniper Moon Farm

Weight	DK (11 wpi) ?
Wraps per inch	
Yardage	273 yards (250 meters)
Unit weight	100 grams (3.53 ounces)
Gauge	21.0 to 24.0 sts = 4 inches
Needle size	US 5 - 7 or 3.75 - 4.5mm
Hook size	4.5mm - 5.5mm (I)
Fibers	100% Wool - Rambouillet

Screenshot of ravelry.com

factor10

```
@Test
fun `calculate total`() {
    val total = YarnCalculator.calculateTotalYardage(200, 5)

    assertEquals(1000, total)
}
```

```
@Test
fun `calculateTotalAvailableYardage for five skeins returns correct yardage`() {
    val skein1 = Skein(Yardage(200, LengthUnit.Meters))

    val sweater = KnittingProject()
        .addSkeins(skein1, 5)

    val actualYardage = sweater.calculateTotalAvailableYardage()

    assertEquals(Yardage(1000, LengthUnit.Meters), actualYardage)
}
```

Lessons from real Projects

factor10

Use tactical patterns to get more comfortable with the domain.

Advantages

- Immutability.
- Type safety.
- Encapsulation of validation and business logic.
- Simple classes.
- Better understandability and testability.
- Explicit domain concepts in code.
- A possibility to start asking questions.
- Accessible entry point into DDD.

Getting Started

- Identify potential Value Objects (primitives that travel together, awkward naming).
- Validate them.
- Make them immutable.
- Implement equality.

Thank you!

katharina.damschen.net

katharina.damschen@factor10.com

linkedIn: katharina-damschen

mastodon.nu/@katharina

Questions?

Just start (with Value Objects)