EAPLI

Um pouco mais sobre o Domínio

Paulo Gandra de Sousa
pag@isep.ipp.pt

DDD Entity as a JPA managed class

```
@Entity
Class Product{
      // database ID
      @ID
      @GeneratedValue(strategy=GenerationType.AUTO)
      private Long id;
      private void setID(Long id) { … }
      private Long getID() { … }

      // domain ID
      private ProductID ref;
      public ProductID getProductID() { … }
      private void setProductID(ProductID ref) { … }

      …
}
```

The need for Value Objects: An example

- Requirement 321:
  - User's passwords must be at least 6 characters longs and have at least one digit

To which class should this responsibility be assigned?

Domain invariants

```
@Test
public void ensurePasswordHasAtLeastOneDigitAnd6CharactersLong() {
        new Password("abcdefgh1");
}

@Test(expected = IllegalArgumentException.class)
public void ensurePasswordsSmallerThan6CharactersAreNotAllowed() {
        new Password("ab1c");
}

@Test(expected = IllegalArgumentException.class)
public void ensurePasswordsWithoutDigitsAreNotAllowed() {
        new Password("abcdefgh");
}
```

You are actually doing design

Domain invariants

```
@Test
public void ensurePasswordHasAtLeastOneDigitAnd6CharactersLong() {
        Password p = Password.valueOf("abcdefgh1");
}

@Test(expected = IllegalArgumentException.class)
public void ensurePasswordsSmallerThan6CharactersAreNotAllowed() {
        Password p = Password.valueOf("ab1c");
}

@Test(expected = IllegalArgumentException.class)
public void ensurePasswordsWithoutDigitsAreNotAllowed() {
        Password p = Password.valueOf("abcdefgh");
}
```

Taking decisions on how the code will work

Domain implementation

```
public class Password {

…

public Password(String password) {
        if (!meetsMinimumRequirements(password)) {
            throw new IllegalStateException();
        }
        thePassword = password;
}

private boolean meetsMinimumRequirements(String password) {
        if (Strings.isNullOrEmpty(password)
        || password.length() < 6)
        || !Strings.containsDigit(password))
        {
            return false;
        }

        return true;
}
```

DDD Value Objects as JPA components

```
@Embeddable
class Color {
      private int red;
      private int green;
      private int blue;
      …
}


@Entity
Class Car {
      …
      private Color color;
      …
}
```

DDD Value Objects as JPA managed classes

```
@Entity
class Color {
        // database ID not to be exposed to domain
        @Id
        @GeneratedValue(strategy=GenerationType.AUTO)
        private int id;

        // domain values
        private int red;
        private int green;
        private int blue;
        …

        // avoid instantiation
        private Color() {...}

        //factory method
        static Color fromRGB(int r, int g, int b) { … }
}


@Entity
Class Car {
        …
        @OneToOne
        private Color color;
        …
}
```

Factory method hides the lookup/write to the DB if necessary.

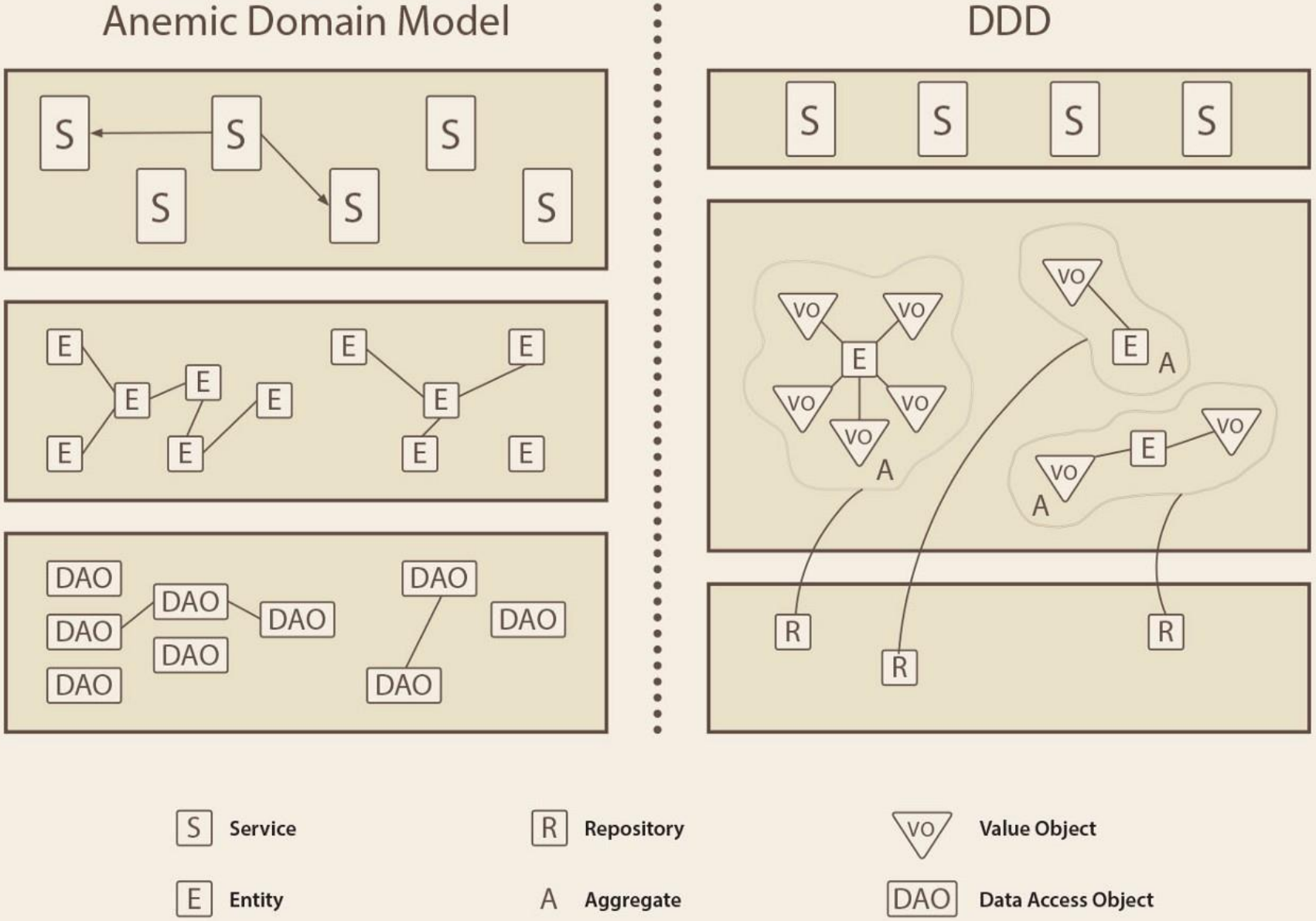
Domain Layer API

All methods of a domain object should handle domain entities and value objects only; no primitive types.

```
void changeName(String name)
```
vs.
```
void changeName(Name aName)
```

Provide a convenience valueOf() method to convert from primitives read from the outside (e.g., UI, DB)

```
Class Name {
      public static Name valueOf(String name) {…}
}
```



https://blog.pragmatists.com/refactoring-from-anemic-model-to-ddd-880d3dd3d45f