

Øvelse 25 – Grundlæggende øvelse med domæneprimitiver

Information

Formålet med denne øvelse er at give en grundlæggende introduktion til domæneprimitiver. Et domæneprimitiv er en klasse, som repræsenterer en attribut (Value Object i DDD) i kildekoden. Nedenstående billede viser en meget simpel domænemodel noteret med UML klassediagramnotation.



I modellen ses det, at personen har fire attributter: fornavn, efternavn, alder og CPR-nummer. Dette betyder, at der i koden skal oprettes fire forskellige klasser for at implementere denne model med domæneprimitiver i koden, altså følgende fire klasser, som skal implementeres:

- Firstname
- Lastname
- Age
- CprNumber

Hver af de fire klasser skal opretholde invariants. For eksempel må alder ikke være under 0, da ingen mennesker kan være under 0 år. Et eksempel er vist herunder:

```
public class Age {
    // Private field that is immutable and an invariant
    private int age;

    // Constructor that accepts an argument and upholds invariance
    public Age(int age) {
        IsAgeValid(age);
        this.age = age;
    }

    // Method to get the value of the Age
    public int GetValue() {
```

```
        return age;
    }

    // Private method to validate age invariant
    private void IsAgeValid(int age) {
        if (age < 0) {
            throw new ArgumentException("Age cannot be less than 0");
        }
    }
}
```

Person-klassen har en alder og skal derfor tage imod et argument, som er af type Age.

```
public class Person {
    // Private field that holds Age object, immutable and upholds invariance
    private Age age;

    // Constructor that accepts Age object and checks for null
    public Person(Age age) {
        if (age == null) {
            throw new ArgumentNullException("Age cannot be null");
        }
        this.age = age;
    }
}
```

Domæneprimitiver understøtter sikkerheden i software ved at sikre mod fejlagtige værdier. For eksempel vil et domæneprimitiv for fornavn, der kun tillader alfabetiske tegn, aldrig tillade et fornavn som f.eks. `<script>You got hacked</script>`.

Alle programmeringssprog har systemprimitiver. For eksempel har C# blandt andet int, bool, char, double og mange flere. Disse understøtter typesikkerhed, således at man ikke kan videregive en komma-værdi som f.eks. `2.4`, hvis typen der skal anvendes er defineret som int (heltal). Dette forhindrer dog ikke, at værdien kan være negativ et sted, hvor det ikke giver mening. For eksempel, hvis alder blot er defineret som en int, kan den sagtens defineres som værende -1. Dette forhindres af domæneprimitiver, ved at definere, hvilken værdi alder må have inden for et givet domæne. Man implementerer altså et typesystem, som understøtter de gældende domænerregler. Dette typesystem sikrer integriteten for de objekter, der eksisterer i softwaren, og øger sikkerheden ved at sikre mod fejlagtige/ondsindede værdier fra f.eks. injection-angreb.

Værktøjer, der anvendes til statisk kodeanalyse, kender ikke til det domæne, som softwaren håndterer. Derfor kan automatiske værktøjer som statisk kodeanalyse ikke finde potentielle sårbarheder i forhold til domænet. Derfor giver domæneprimitiver et ekstra lag af sikkerhed.

Ydermere kan man oprette unit tests til domæneprimitiver (hvis du ikke ved, hvad en unit test er, kommer det senere i undervisningen), hvilket sikrer, at domænerregler ikke uhensigtsmæssigt bliver overtrådt, når der foretages ændringer i kildekoden.

□ Instruktioner

1. □ Opret en ny .NET-konsolapplikation.
 2. □ Opret et domæneprimitiv for **fornavn**. Fornavn må kun indeholde alfabetiske tegn og skal have en længde på mellem 2 og 20 karakterer.
 3. □ Opret et domæneprimitiv for **efternavn**. Efternavn må kun indeholde alfabetiske tegn og skal have en længde på mellem 2 og 20 karakterer.
 4. □ Opret et domæneprimitiv for **alder**. Alderen skal være mellem 18 og 99.
 5. □ Opret et domæneprimitiv for **CPR-nummer**. CPR-nummeret skal overholde syntaksen for et dansk CPR-nummer (kun syntaksen, køn kan du undlade).
 6. □ Opret en `GetValue`-metode i alle domæneprimitiver, således at værdien kan udlæses (hvis du ikke allerede har gjort det).
 7. □ Opret en **Person-klasse**, der indeholder et private field for hver af de fire domæneprimitiver.
 8. □ Lad konstruktøren i Person-klassen tage imod et argument for hver af de fire domæneprimitiver og brug argumenterne til at initialisere de private fields.
 9. □ Lav en `GetFirstname`, `GetLastname`, `GetAge`, `GetCprNumber`-metode, og lad dem returnere værdien af domæneprimitivet (husk at bruge `GetValue`-metoden).
 10. □ Opret et Person-objekt i konsolapplikationen og udskriv alle fire værdier.
 11. □ (Ekstra) CPR-nummer er følsomme data. Implementer domæneprimitivet til CPR-nummer med *Read Once Pattern* (se afsnit 5.2 i bogen *Secure by Design*).
 12. □ (Ekstra) Lav invariansen i domæneprimitivet til CPR-nummer, således at CPR-nummeret passer med personens køn.
-
-

□ Refleksionsspørgsmål

1. Hvordan kan brugen af domæneprimitiver bidrage til at sikre bedre softwarekvalitet og robusthed?
 2. Hvorfor er immutability (at objekter ikke kan ændres efter oprettelse) vigtig i forhold til software-sikkerhed?
 3. Hvilke typer fejl og sikkerhedsproblemer kan forebygges ved at håndhæve invarians i domæneprimitiver?
 4. Hvordan kan man balancere brugen af domæneprimitiver for at sikre robusthed uden at gøre koden unødvendigt kompleks eller svær at vedligeholde?
-

□ Kobling til relevante standarder og principper

Element i øvelsen	Relateret standard / princip	Kommentar
Invariants håndhævelse	ISO/IEC 27002:2022, afsnit 8.26 (applikationssikkerhed)	Anbefaler validering og integritetstjek af input til en applikation.
	OWASP ASVS 5.0.0, Kontrol 16.5 (Error Handling)	Krav om korrekt håndtering af fejl(undtagelser)
	Secure by Design (Johnsson et al.)	Fremhæver princippet om at skabe robuste domænemodeller og sikre objekters tilstand gennem konstruktion.

□ Domæneprimitiver hjælper med at sikre, at kun gyldige data kan eksistere i systemet, hvilket forebygger mange almindelige fejl og sårbarheder.

Last update: 2025-10-30 12:07:19