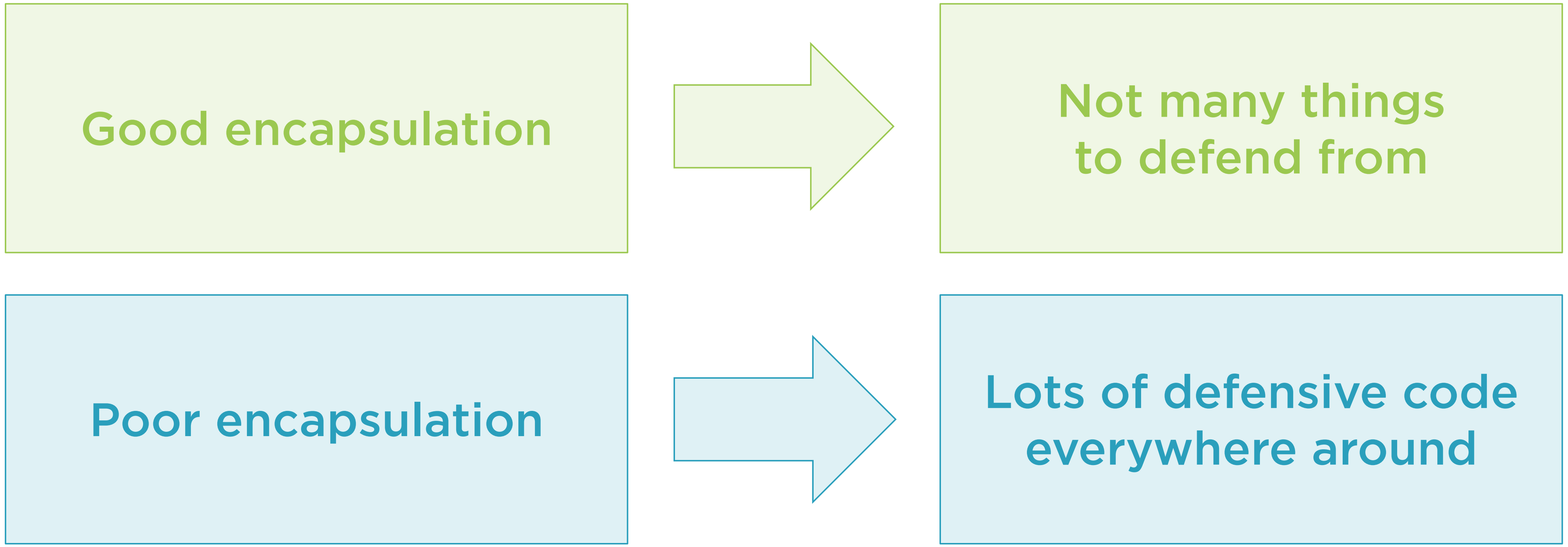
Building Defensive Design Instead of Writing Defensive Code

Zoran Horvat
PRINCIPAL CONSULTANT AT CODING HELMET

@zoranh75 csharpmentor.com

Encapsulation and Defense


Good encapsulation
Not many things to defend from
Poor encapsulation
Lots of defensive code everywhere around

Encapsulation defined

- A language mechanism for restricting direct access to some of the object's components.

State

- A language construct that facilitates the bundling of data with the methods (or other functions) operating on that data.

Operations (methods)

https://en.wikipedia.org/wiki/Encapsulation_(computer_programming)

Encapsulating State

```
class Student
{
  private string lastName;

  public string LastName
  {
    get { return lastName; }
    set
    {
      if (string.IsNullOrEmpty(value))
        throw new ArgumentException();
      lastName = value;
    }
  }
}
```

Caller will get the student's last name and then do all the work itself!

Invalid state cannot find its way to the private field

Encapsulating State

```
class Student
{
  private string lastName;

  public string LastName
  {
    get { return lastName; }
    set
    {
      if (string.IsNullOrEmpty(value))
        throw new ArgumentException();
      lastName = value;
    }
  }
}
```

Consequence

When only **state** is encapsulated, then **behavior** is put far away from the data.

Encapsulating State

```
class Student
{
  private string lastName;

  public string LastName
  {
    get { return lastName; }
    set
    {
      if (string.IsNullOrEmpty(value))
        throw new ArgumentException();
      lastName = value;
    }
  }
}
```

Example

LINQ query to filter students by their initial

```
students.Where(student =>
  student.LastName.StartsWith('S');
```

Sample data	
Last name	**Initial letter**
Smith	S
Simpson	S
O'Sullivan	O or S?
de Smedt	D or S?
De Smedt	D or S?

Encapsulating Behavior

```
class Student
{
  ...
  public bool LastNameStartsWith(char letter)
  {
    int index = 0;

    if (this.lastName.StartsWith("O'"))
      index = 2;

    if (this.lastName.ToLower().StartsWith("de "))
      index = 3;

    return
        this.lastName[index].ToLower() ==
        letter.ToLower();
  }
}
```

Support for traditional family names is encapsulated

Case-insensitiveness is encapsulated

Encapsulating State vs. Encapsulating Behavior

With encapsulated state:

```
students.Where(student => student.LastName.StartsWith('S')); ✗
```

With encapsulated behavior:

```
students.Where(student => student.LastNameStartsWith('S')); ✓
```

Last name	Initial (encapsulated state)	Initial (encapsulated behavior)
Smith	S ✓	S ✓
Simpson	S ✓	S ✓
O'Sullivan	O ✗	S ✓
de Smedt	D ✗	S ✓
De Smedt	D ✗	S ✓

Encapsulating State vs. Encapsulating Behavior

With encapsulated state:

```
students.Where(student => student.LastName.StartsWith('S')); ✗
```

With encapsulated behavior:

```
students.Where(student => student.LastNameStartsWith('S')); ✓
```

Encapsulation defined

A language construct that facilitates the bundling of data with the methods (or other functions) operating on that data.

Wikipedia

Encapsulating State vs. Encapsulating Behavior

```
class Student
{
  private string LastName { get; }
  public bool LastNameStartsWith(char letter);
  public bool LastNameContains(string part);
  public int CompareLastNameWith(Student other);
  public bool IsMultipartLastName();
}
```

Encapsulating State vs. Encapsulating Behavior

```
class Student
{
  private string LastName { get; }
  ~~public bool LastNameStartsWith(char letter);~~
  ~~public bool LastNameContains(string part);~~
  ~~public int CompareLastNameWith(Student other);~~
  ~~public bool IsMultipartLastName();~~
  ~~...~~
}
```

Encapsulation defined

A language mechanism for restricting direct access to some of the object's components.

Wikipedia

Encapsulating State vs. Encapsulating Behavior

```
class Student
{
  private string LastName { get; }
  public char LastNameInitial { get { ... } }
}
```

Raw, unprocessed state remains *private*

New property is a *projection* of the encapsulated state!

Behavior is encapsulated in the feature provider

Encapsulation defined

A language mechanism for restricting direct access to some of the object's components.

Wikipedia

Encapsulating State vs. Encapsulating Behavior

```
class Student
{
  private string LastName { get; }
  public char LastNameInitial { get { ... } }
}
```

Raw, unprocessed state remains *private*

New property is a *projection* of the encapsulated state!

Behavior is encapsulated in the feature provider

```
students.Where(student => student.LastNameInitial.ToLower() == ‘S’)...
```

Producer tells the initial

Consumer says letter casing is not important

Good separation of responsibilities

Encapsulating State vs. Encapsulating Behavior

```
class Student
{
  private string LastName { get; }
  public char LastNameInitial { get { ... } }
  public int CompareLastNameWith(Student other);
}
```

Keep raw data private

Expose processed data only

Project state into a generally useful form

Method returning a value is also projecting private state, in a way

Advice

Strike the right balance between only exposing *state* and only exposing *behavior*

Encapsulation and Cohesion

```
class Student
{
  private string LastName { get; }
  public char LastNameInitial
  {
    get
    {
      Logic is encapsulated here
    }
  }
}
```

Encapsulation and Cohesion

Definition of cohesion

- Keeping related things together

Student's class responsibilities

- Enlist for semesters
- Take labs and exams
- Collect grades
- Know traditional family names

Cohesion in the Student Class

```
class Student
{
  private string lastName;

  public char LastNameInitial
  {
    get
    {
      int index = 0;

      if (this.lastName.StartsWith("O'"))
        index = 2;

      if (this.lastName.ToLower().StartsWith("de "))
        index = 3;

      return this.lastName[index];
    }
  }
}
```

False assumption

Defensive code

Concrete cases

Cohesion in the Student Class

```
class Student
{
  private string lastName;

  public char LastNameInitial
  {
    get
    {
      int index = 0;

      if (this.lastName.StartsWith("O'"))
        index = 2;

      if (this.lastName.ToLower().StartsWith("de "))
        index = 3;

      return this.lastName[index];
    }
  }
}
```

Encapsulates access to private state

Cohesion in the Student Class

```
class Student
{
    private string lastName;

    public char LastNameInitial { get { ... } }

    public void SetGrade(Grade grade) { ... }
    public void Enlist(Semester semester) { ... }
    public void TakeExamOn(Subject subject) { ... }
    ...
}
```

Unrelated members can access this field as if it were public!

Low cohesion:

Class breaks its own encapsulation.

Cohesion in the Student Class

```
class Student
{
  private string lastName;

  private Semester enlisted;
  private List<Grade> grades;
  private List<Subject> takenExams;

  public char LastNameInitial { get { ... } }

  public void SetGrade(Grade grade) { ... }
  public void Enlist(Semester semester) { ... }
  public void TakeExamOn(Subject subject) { ... }
  ...
}
```

Advice

Class with low cohesion should be split into two.

Introducing the Chain of Responsibility

```
public char LastNameInitial
{
  get
  {
    int index = 0;
    if (this.LastName.StartsWith("O'"))
      index = 2;
    if (this.LastName.ToLower().StartsWith("de "))
      index = 3;
    return this.LastName[index];
  }
}
```

Issue #1:
Adding more cases requires code change

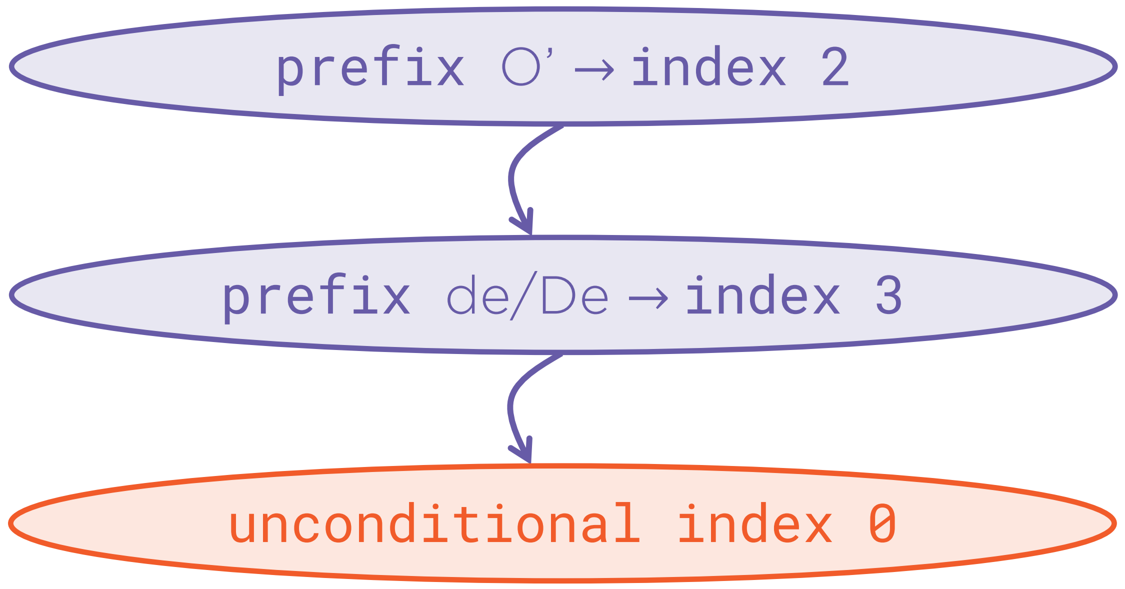
Issue #2:
Incomplete branching logic causes bugs

Issue #3:
Branching can easily be misunderstood, causing bugs

Introducing the Chain of Responsibility

```
public char LastNameInitial
{
  get
  {
    int index = 0;
      prefix O' → index 2
      prefix de/De → index 3
    return this.LastName[index];
  }
}
```

Introducing the Chain of Responsibility

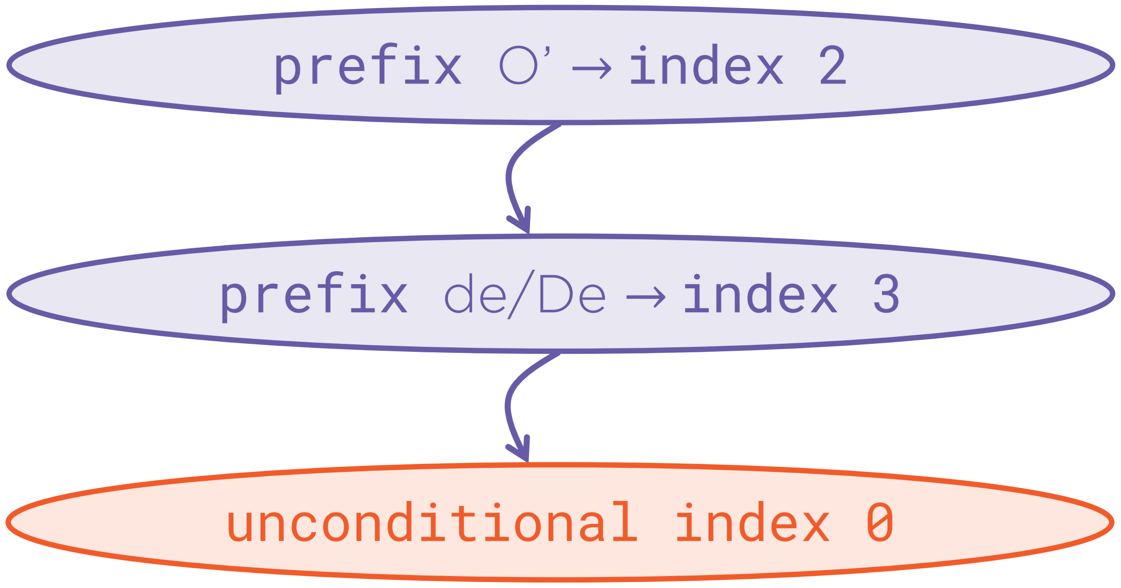


Chain of Responsibility
Request is passed down the chain until served.

Catch-all element
Terminates the chain and processes all requests.

Introducing the Chain of Responsibility

Constructed at run time

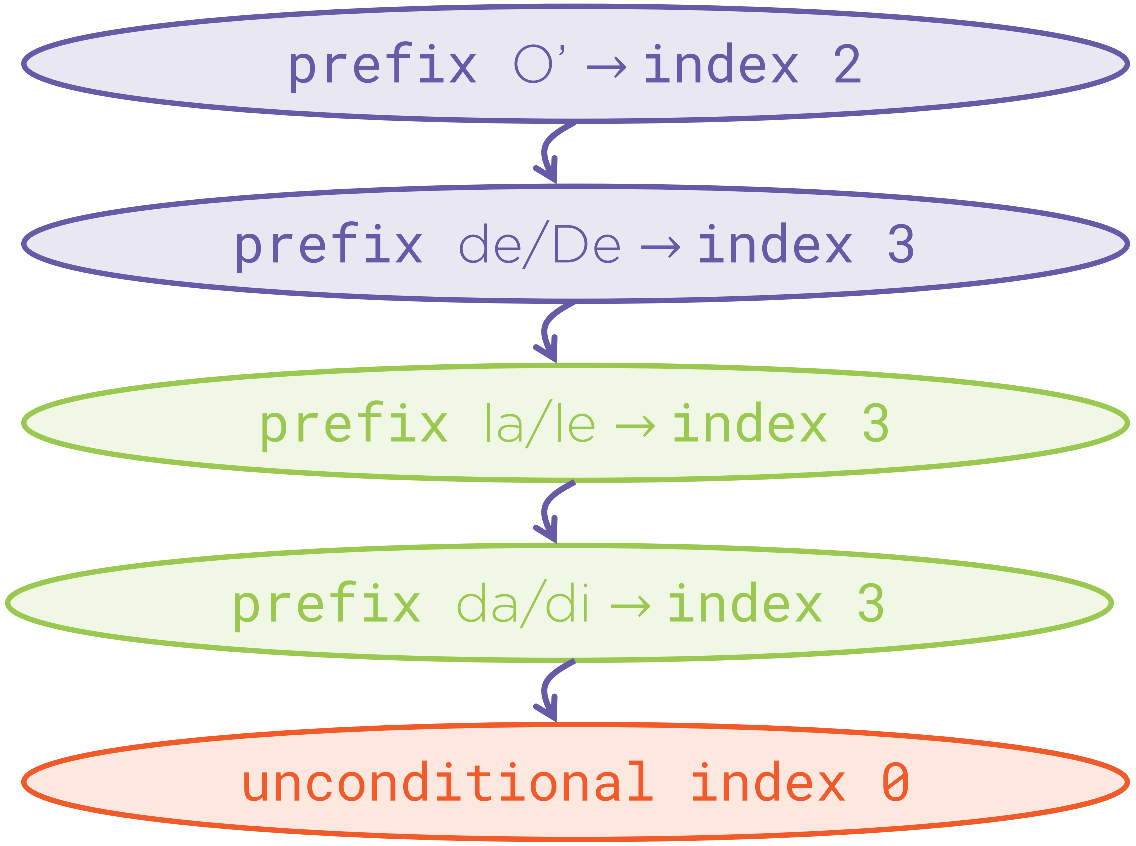


Constructed at compile time

```
if (n.StartsWith("O'"))
  index = 2;
else if (n.ToLower().StartsWith("de "))
  index = 3;
else
  index = 0;
```

Introducing the Chain of Responsibility

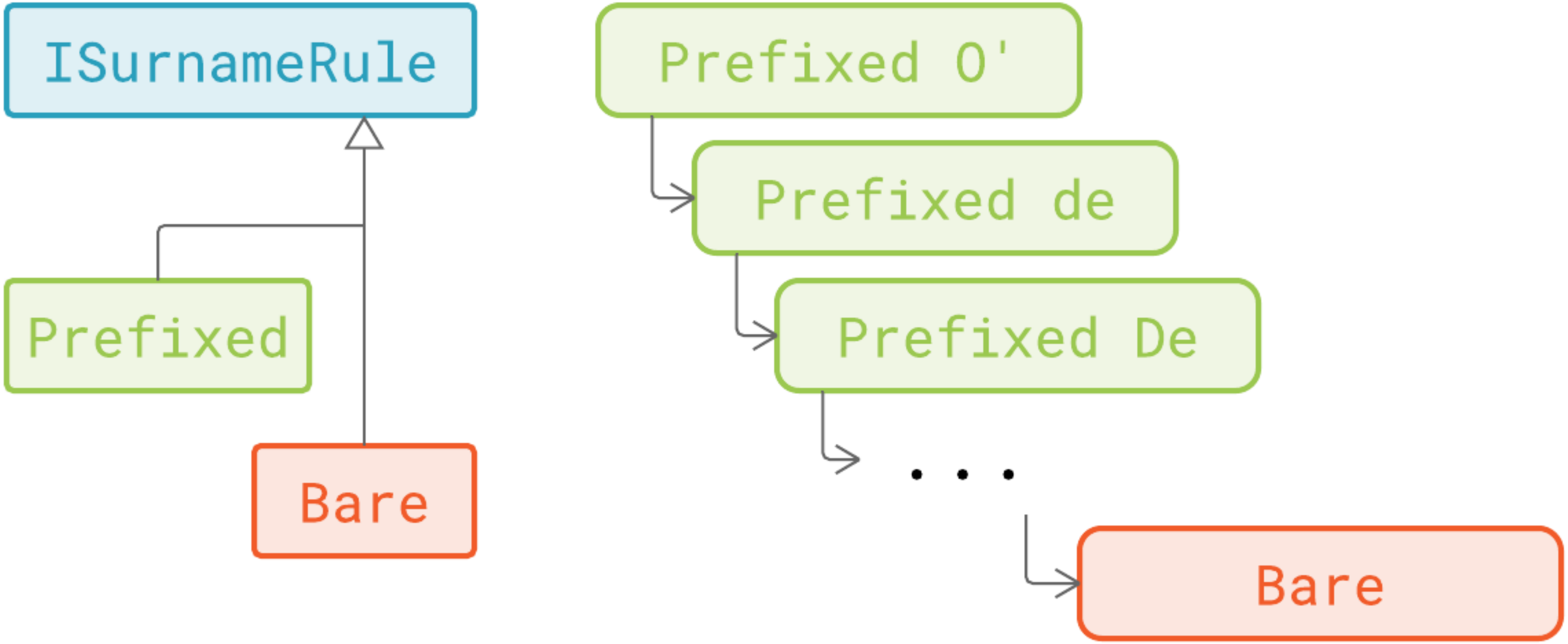
Constructed at run time



Constructed at compile time

```
if (n.StartsWith("O'"))
  index = 2;
else if (n.ToLower().StartsWith("de "))
  index = 3;
else if (n.StartsWith("la "))
  index = 3;
else if (n.StartsWith("le "))
  index = 3;
else if (n.StartsWith("da "))
  index = 3;
else if (n.StartsWith("di "))
  index = 3;
else
  index = 0;
```

Introducing the Chain of Responsibility



Benefits of Using Regular Expressions

Expressions, not code

Domain-Specific Language (DSL)

Configurable

Testable

Expressions are not executable

Expressions are plain text

Benefits of Using Regular Expressions

Expressions, not code

Domain-Specific Language (DSL)

Configurable

Testable

Serve as text processing language

No need for general-purpose code

Benefits of Using Regular Expressions

Expressions, not code

Domain-Specific Language (DSL)

Configurable

Testable

Keep expressions separate from code

Pull them from configuration, database, file, etc.

Benefits of Using Regular Expressions

Expressions, not code

Domain-Specific Language (DSL)

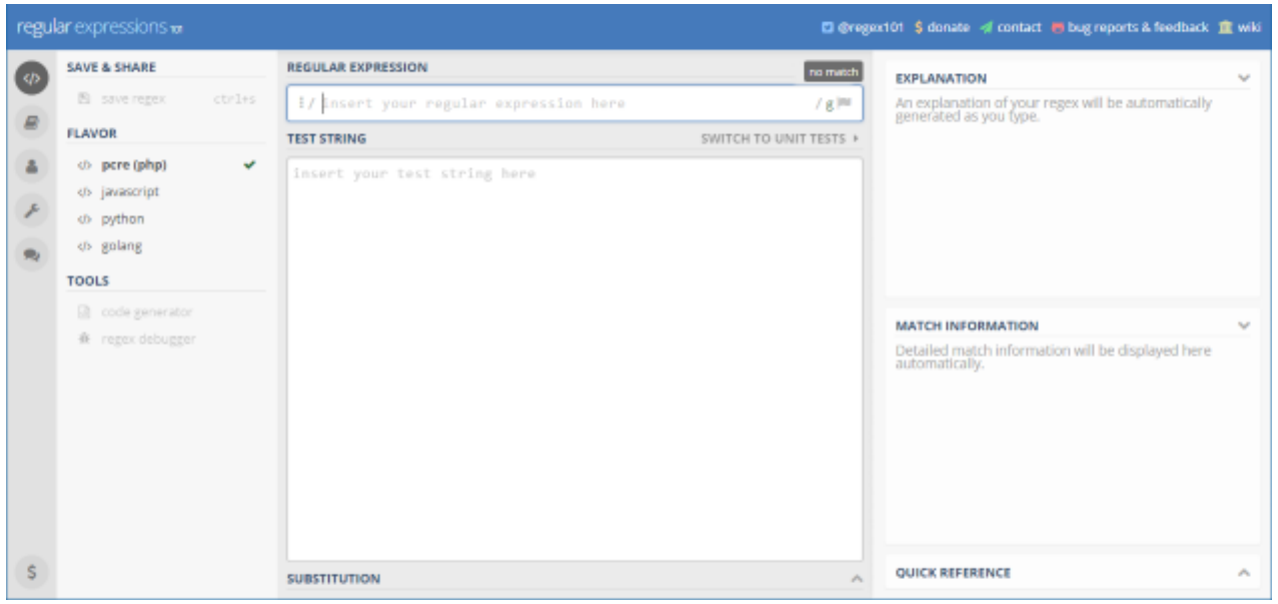
Configurable

Testable

Try them in a stand-alone emulator

Plenty of online testers available

https://regex101.com



Summary

Encapsulation

- Validation before modifying state
- Keeping operations close to data

Encapsulating behavior

- Domain rules and logic located together with the affected data

Operations have nothing to defend from when located close to correct data

- There will be no exceptions at run time

Summary

Alternative computation models

- Rules instead of step-by-step process

Dealing with rules

- Externalize them into a separate type
- Hosting object only triggers the rules
- Rules become an object’s dependency

Summary

Chain of Responsibility pattern

- Refer to *Tactical Design Patterns in .NET: Managing Responsibilities* course

Regular expressions

- Organized into a list to form a Chain
- Great match for text processing tasks

After externalizing rules

- Class has nothing to defend from (except against null references)

Next module

Working with Objects, Not with Nulls