Datentypen, die den Definitionsbereich genau abbilden

Adrian Imboden

adi@thingdust.com
adrian.imboden@komaxgroup.com

`https://github.com/adrianimboden/cppusergroup-domain-driven-datatypes`

October 15, 2019

Inhalt

Intro

Motivation

Basics

Challenge für die, die wollen

Hands-On

Inhalt

Intro

Motivation

Basics

Challenge für die, die wollen

Hands-On

Intro

- Wir haben genug Zeit
- Der Hauptteil wird Hands-On sein
- Diskussionen sind explizit erwünscht

Inhalt

Intro

Motivation

Basics

Challenge für die, die wollen

Hands-On

Motivation

```
my_list = [1, 2, 3]
do_something(my_list)
assert (my_list == [1, 2, 3])
```

Motivation

```python
secret_global = None


def do_something(value):
    global secret_global
    value.append('new fancy value')  # no const
    secret_global = value  # everything is shared


my_list = [1, 2, 3]
do_something(my_list)
assert (my_list == [1, 2, 3])
```

Motivation

```java
import java.util.ArrayList;

public class HelloWorld
{
  static ArrayList<Integer> cache;
  static void doSomething(ArrayList<Integer> list) { // null?
    list.add(5);  // no language-level const
    cache = list; // (almost) everything is always shared
  }

  public static void main(String[] args)
  {
    final ArrayList<Integer> my_list = new ArrayList<>();
    my_list.add(1);
    doSomething(my_list);
    doSomething(null);
  }
}
```

Motivation

```
using System;
using System.Collections.Generic;

public class HelloWorld
{
  public static List<int> cache;
  public static void doSomething(List<int> list) { //null?
    list.Add(5);  // no language-level const
    cache = list; // shareability is a type-design decision
  }

  public static void main(String[] args)
  {
    List<int> my_list = new List<int>();
    my_list.Add(1);
    doSomething(my_list);
    doSomething(null);
  }
}
```

Inhalt

Intro

Motivation

Basics

Challenge für die, die wollen

Hands-On

Value Type

Alles was sich wie ein “int” verhält:

- Kopierbar
- Kein (sichtbares) sharing
- Keine Werte ausserhalb vom Wertebereich (nicht nullable)
- const ist transitiv

Value Type

Alles was sich wie ein “int” verhält:

- Kopierbar
- Kein (sichtbares) sharing
- Keine Werte ausserhalb vom Wertebereich (nicht nullable)
- const ist transitiv

Konkret:

- int
- std::string
- std::vector
- std::optional
- std::variant

Value Type

Alles was sich wie ein “int” verhält:

- Kopierbar
- Kein (sichtbares) sharing
- Keine Werte ausserhalb vom Wertebereich (nicht nullable)
- const ist transitiv

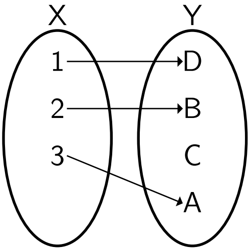
Konkret:

- int
- std::string
- std::vector
- std::optional
- std::variant

Kurz: fast alle Typen in der Standardlibrary

Definitionsmenge

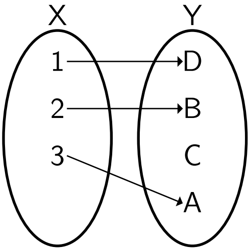
In der Mathematik versteht man unter Definitionsmenge oder Definitionsbereich die Menge mit genau den Elementen, unter denen die Funktion definiert bzw. die Aussage erfüllbar ist.



Quelle: wikipedia.org

Definitionsmenge

In der Mathematik versteht man unter Definitionsmenge oder Definitionsbereich die Menge mit genau den Elementen, unter denen die Funktion definiert bzw. die Aussage erfüllbar ist.



Quelle: wikipedia.org

```
using X = int;
using Y = char;

Y foo(X input) {
  switch (input) {
  case 1: return 'D';
  case 2: return 'B';
  case 3: return 'A';
  }
  throw std::out_of_range{
    "value out of domain",
  };
}
```

Datentyp für Definitionsmenge

```
class X {
  int value_;

public:
  explicit X(int value) : value_{value} {
    if ((value_ < 1) or (value_ > 3))
      throw std::out_of_range{
        "value not in domain",
      };
  }

  operator int() const { return value_; }
};
```

Vergleich

```
using X = int;
```

```
Y foo(X input) {
  switch (input) {
  case 1: return 'D';
  case 2: return 'B';
  case 3: return 'A';
  }
  throw std::out_of_range{
    "value out of domain",
  };
}

int main() {
  const auto value = X{5};
  foo(value); // throws
}
```

Vergleich

```
using X = int;
```

```
class X {
public:
  explicit X(int value);
  operator int() const;
};
```

```
Y foo(X input) {
  switch (input) {
  case 1: return 'D';
  case 2: return 'B';
  case 3: return 'A';
  }
  throw std::out_of_range{
    "value out of domain",
  };
}

int main() {
  const auto value = X{5};
  foo(value); // throws
}
```

Vergleich

```
using X = int;
```

```
Y foo(X input) {
  switch (input) {
  case 1: return 'D';
  case 2: return 'B';
  case 3: return 'A';
  }
  throw std::out_of_range{
    "value out of domain",
  };
}

int main() {
  const auto value = X{5};
  foo(value); // throws
}
```

```
class X {
public:
  explicit X(int value);
  operator int() const;
};
```

```
Y foo(X input) {
  switch (input) {
  case 1: return 'D';
  case 2: return 'B';
  case 3: return 'A';
  }
  assert(false); // unreachable
}
```

Vergleich

```
using X = int;




Y foo(X input) {
  switch (input) {
  case 1: return 'D';
  case 2: return 'B';
  case 3: return 'A';
  }
  throw std::out_of_range{
    "value out of domain",
  };
}

int main() {
  const auto value = X{5};
  foo(value); // throws
}
```

```
class X {
public:
  explicit X(int value);
  operator int() const;
};

Y foo(X input) {
  switch (input) {
  case 1: return 'D';
  case 2: return 'B';
  case 3: return 'A';
  }
  assert(false); // unreachable
}

int main() {
  const auto value = X{5}; // throws
  foo(value);
}
```

Vergleich

```cpp
int roll_dice() {
  //...
  return 8;
}

int yazzee_count(std::array<int, 5> dies) {
  //...
  return 5;
}

int main() { //
  yazzee_count({
    roll_dice(),
    roll_dice(),
    roll_dice(),
    roll_dice(),
    roll_dice(),
  });
}
```

Vergleich

```
class DieRoll {
  int value_;

public:
  explicit DieRoll(int value) : value_{value} {
    if (value < 1 or value > 6)
      throw std::invalid_argument{"invalid roll"};
  }
  operator int() { return value_; }
};
```

```
int roll_dice() {
  //...
  return 8;
}

int yazzee_count(std::array<int, 5> dies) {
  //...
  return 5;
}

int main() { //
  yazzee_count({
    roll_dice(),
    roll_dice(),
    roll_dice(),
    roll_dice(),
    roll_dice(),
  });
}
```

Vergleich

```cpp
int roll_dice() {
  //...
  return 8;
}

int yazzee_count(std::array<int, 5> dies) {
  //...
  return 5;
}

int main() { //
  yazzee_count({
    roll_dice(),
    roll_dice(),
    roll_dice(),
    roll_dice(),
    roll_dice(),
  });
}
```

```cpp
class DieRoll {
  int value_;

public:
  explicit DieRoll(int value) : value_{value} {
    if (value < 1 or value > 6)
      throw std::invalid_argument{"invalid roll"};
  }
  operator int() { return value_; }
};

DieRoll roll_dice() {
  //...
  return DieRoll{7};
}

int yazzee_count(std::array<DieRoll, 5> dies) {
  //...
  return 5;
}

int main() { //
  yazzee_count({
    roll_dice(),
    roll_dice(),
    roll_dice(),
    roll_dice(),
    roll_dice(),
  });
}
```

Realbeispiele aus der Praxis

- Utf8String
- Message
- Milliesconds
- Configuration

Inhalt

Intro

Motivation

Basics

Challenge für die, die wollen

Hands-On

Beispiel Zahlen

```cpp
template <typename T, T from, T to>
class Number {
  T value_;

public:
  static_assert(from <= to);
  explicit Number(T value)
    : value_{value} {
    if (!((value_ >= from) && (value_ <= to))) {
      throw std::out_of_range("value not in domain");
    }
  }

  operator T() const { return value_; }
};

int main() {
  const auto a = Number<int, 2, 5>{3};
  const auto b = Number<int, 1, 2>{2};

  const auto c = Number<int, 4, 5>{a}; //-> runtime error
  const auto d = Number<int, 1, 2>{8}; //-> runtime error
  const auto e = Number<int, 1, 5>{a}; //-> implicit conversion
}
```

Beispiel Zahlen Erweitert

```cpp
int main() {
  const auto a = Number<int, 2, 5>{3};
  const auto b = Number<int, 1, 2>{2};

  static_assert(std::is_same_v<decltype(a + b), Number<int, 3, 7>>);
  static_assert(std::is_same_v<decltype(a - b), Number<int, 0, 4>>);
  static_assert(std::is_same_v<decltype(a * b), Number<int, 2, 10>>);
  static_assert(std::is_same_v<decltype(a / b), Number<int, 1, 5>>);

  constexpr auto c = Number<int, 1, 2>{8}; //-> does not compile
  const auto d = Number<int, 1, 2>{8}; //-> runtime error
  const auto e = Number<int, 4, 5>{a}; //-> static assert
  const auto f = Number<int, 1, 5>{a}; //-> implicit conversion
}
```

Links

- Boost.TypeErasure (`https://www.boost.org`)
- CppCon 2018 Talks
  - Value Semantics: Fast, Safe and Correct (`https://www.youtube.com/watch?v=PkyD1iv3ATU`)
  - How to Write Well-Behaved Value Wrappers (`https://www.youtube.com/watch?v=J4A2B9eexiw`)

Inhalt

Intro

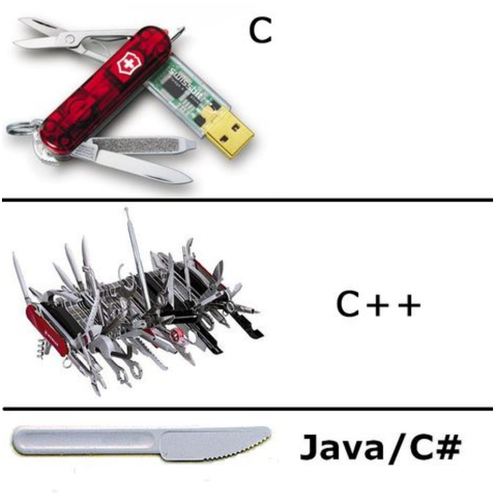
Motivation

Basics

Challenge für die, die wollen

Hands-On

Hands-on



Quelle: https://www.pinterest.com

The End