Schools of TDD

@thbrunzendorf
@MarcoEmrich

codecentric

ABOUT TDD

WHEN?

WHO?

1957

INVENTED

JOHN VON NEUMAN

1989

REDISCOVERED

KENT BECK

30 YEARS OF TDD

…OR 60 ?

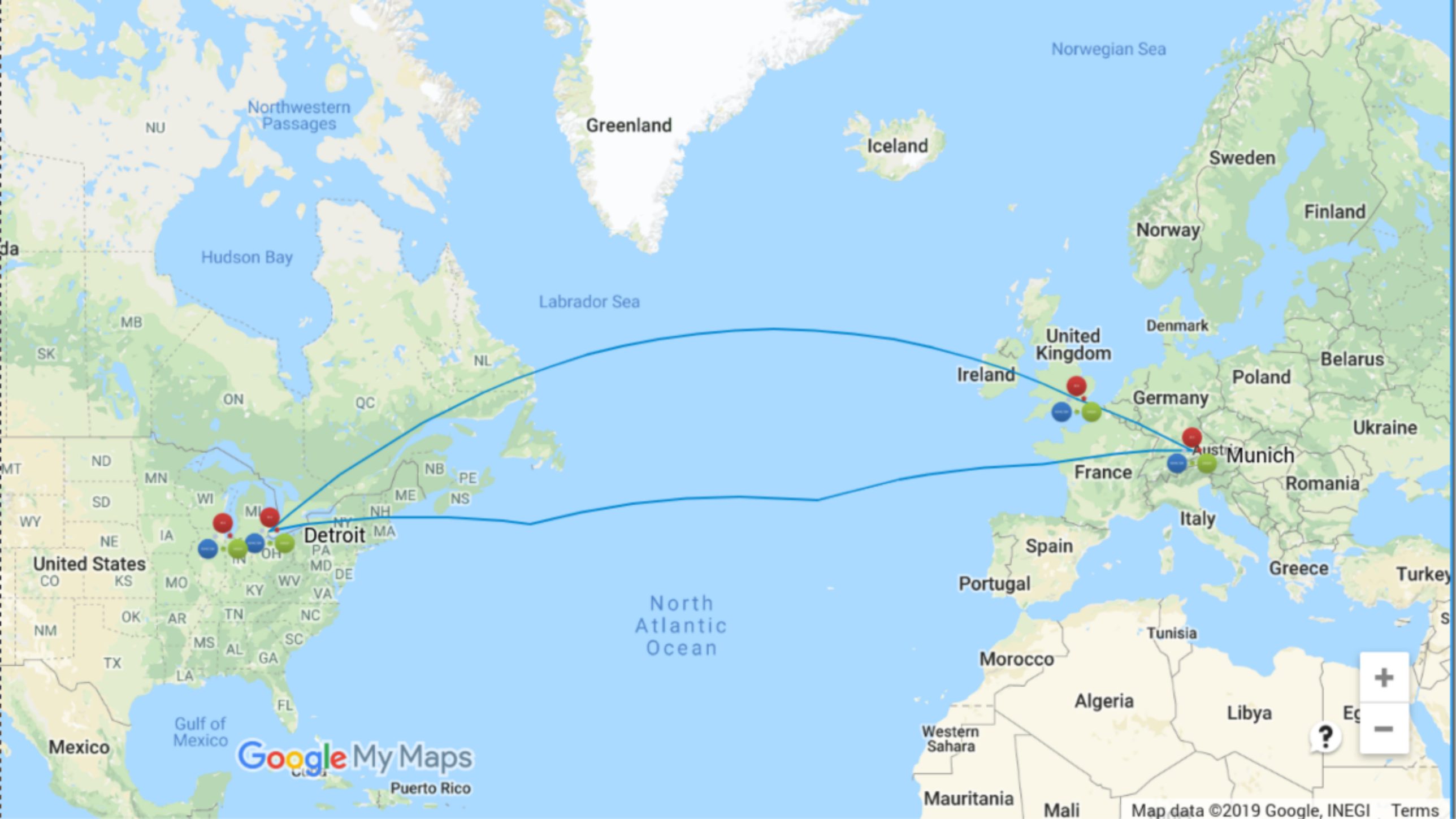
WHAT HAPPENED IN THESE YEARS?

IMPROVING TDD-SKILLS TODAY?


SCHOOLS...
NEPAL

> *Martial arts can be grouped by type or focus, or alternatively by* ***regional origin****.*

— Wikipedia: List_of_martial_arts

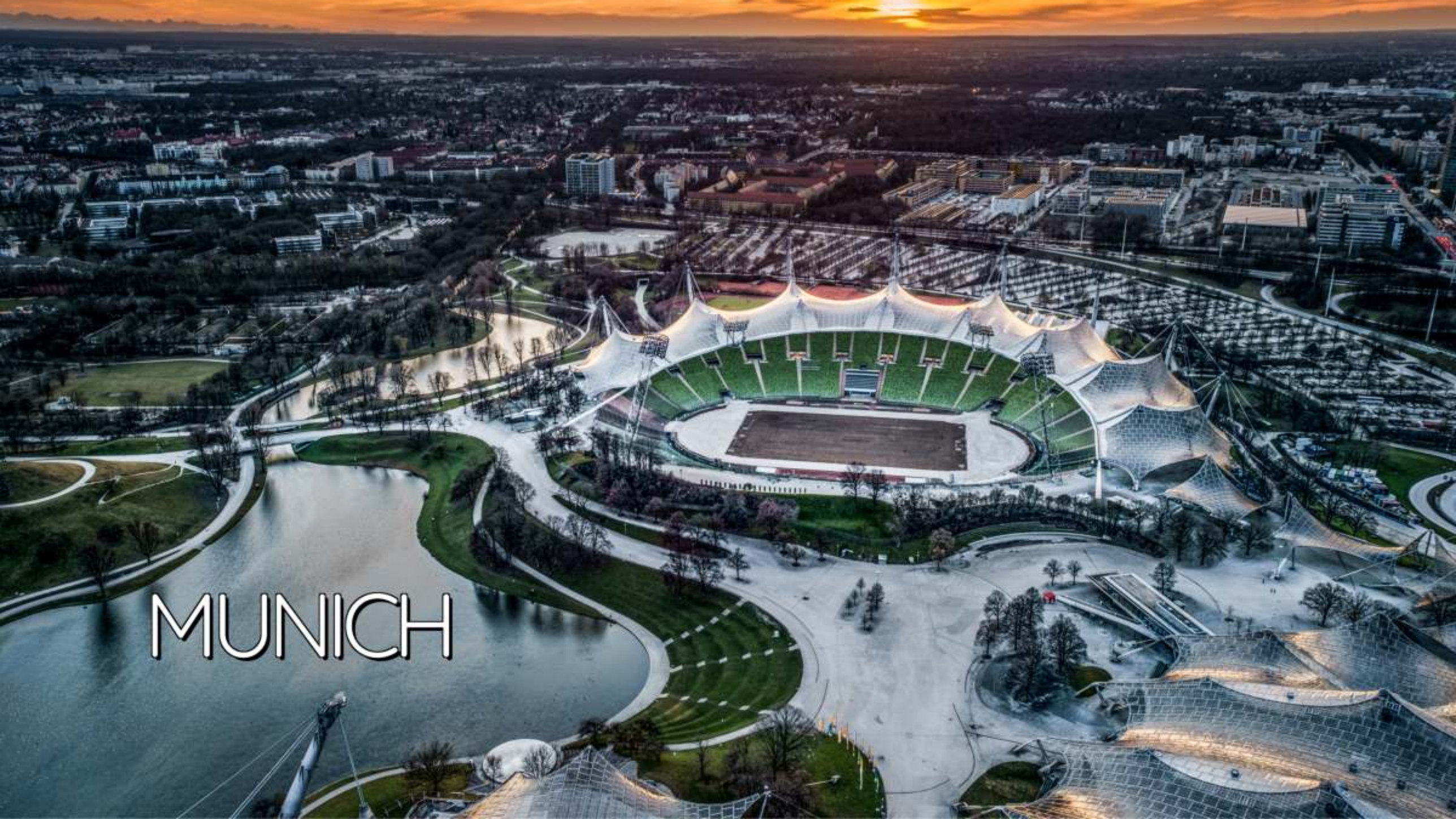

Norwegian Sea
Northwestern Passages
NU
Greenland
Iceland
Sweden
Finland
Norway
Hudson Bay
Labrador Sea
MB
SK
ON
QC
NL
Denmark
United Kingdom
Ireland
Belarus
Poland
Germany
Ukraine
Munich
France
Romania
Italy
Spain
Portugal
Greece
Turkey
Tunisia
Morocco
Algeria
Libya
Western Sahara
Mauritania
Mali
ND
MN
SD
WI
MI
WY
NE
IA
IL
IN
OH
NY
PA
NB
PE
NS
ME
NH
MA
MD
DE
WV
VA
KY
NC
TN
SC
MS
AL
GA
FL
LA
TX
OK
AR
NM
CO
KS
MO
MT
Detroit
United States
North Atlantic Ocean
Gulf of Mexico
Mexico
Puerto Rico
Google My Maps
Map data ©2019 Google, INEGI
Terms


CHICAGO

KENT BECK


LONDON

NAT PRICE & STEVE FREEMAN


MUNICH

DAVID VÖLKEL


TERMS
&
BUILDING BLOCKS

SUBJECT UNDER TEST

SUBJECT UNDER TEST: SUT

- Object
- Function / Method
- Module
- Software System
- ... something else?


TEST DOUBLES
MOCKS/SPIES
STUBS

ASSERTIONS & EXPECTATIONS

FRONTDOOR TESTING

VS

BACKDOOR TESTING

FRONTDOOR TESTING

- Result Verification
- State Verification

RESULT VERIFICATION

```
test('add adds up two numbers', () => {
  expect( add(3, 4) ).toEqual(7);
});
```

STATE VERIFICATION

```
test('addCardOnTop should increase number of Cards in Deck', () => {
  deck = new DeckOfCards();
  deck.addCardOnTop("♥9")
  expect(deck.numberOfCards).toEqual(1);
  expect(deck.topCard).toEqual("♥9");
});
```

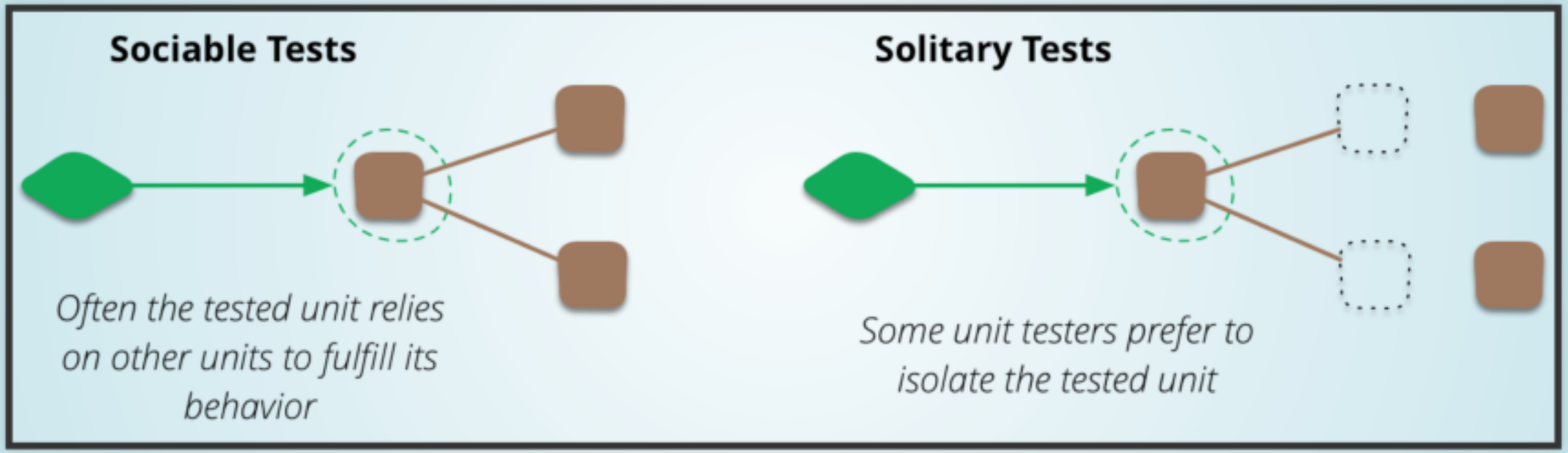
BACKDOOR TESTING

- Behavior verification
- Test Doubles: Mocks & Spys

BEHAVIOR VERIFICATION

```
test('addCardOnTop should call shuffle on collection', () => {
  const collectionSpy = {randomizeOrder: jest.fn()};
  deck = new DeckOfCards(collectionSpy);
  deck.shuffle();
  expect(collectionSpy.randomizeOrder).toHaveBeenCalled();
});
```

SOCIABLE VS SOLITARY*



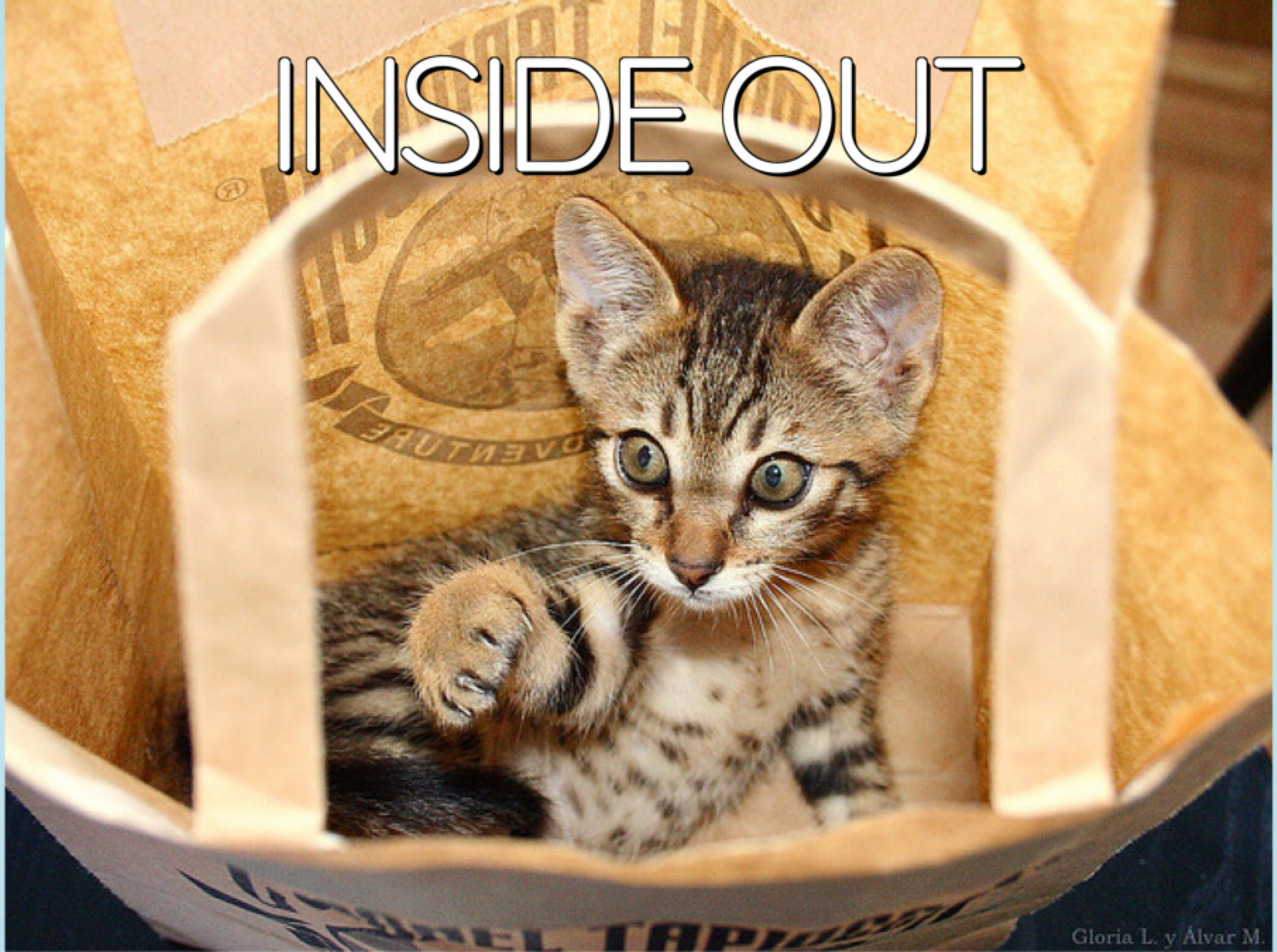
(*) stolen from M. Fowler

INSIDE OUT

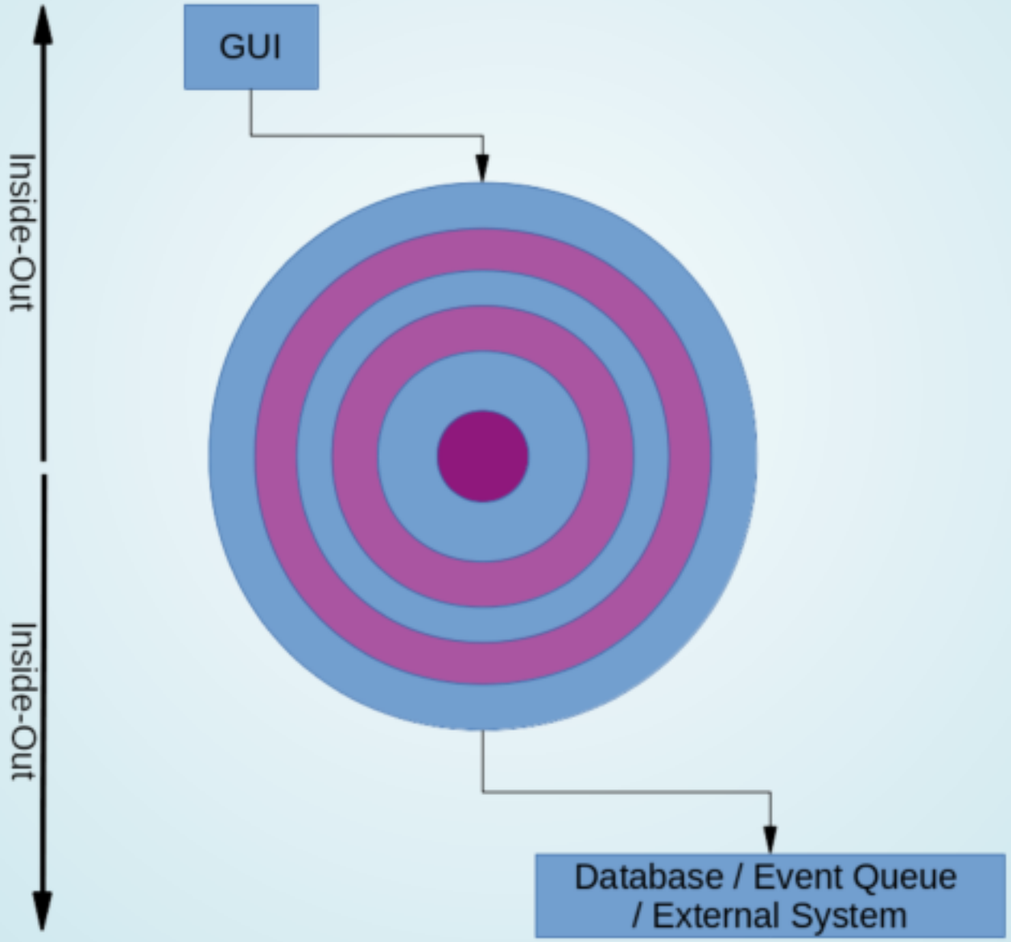
VS

OUTSIDE IN

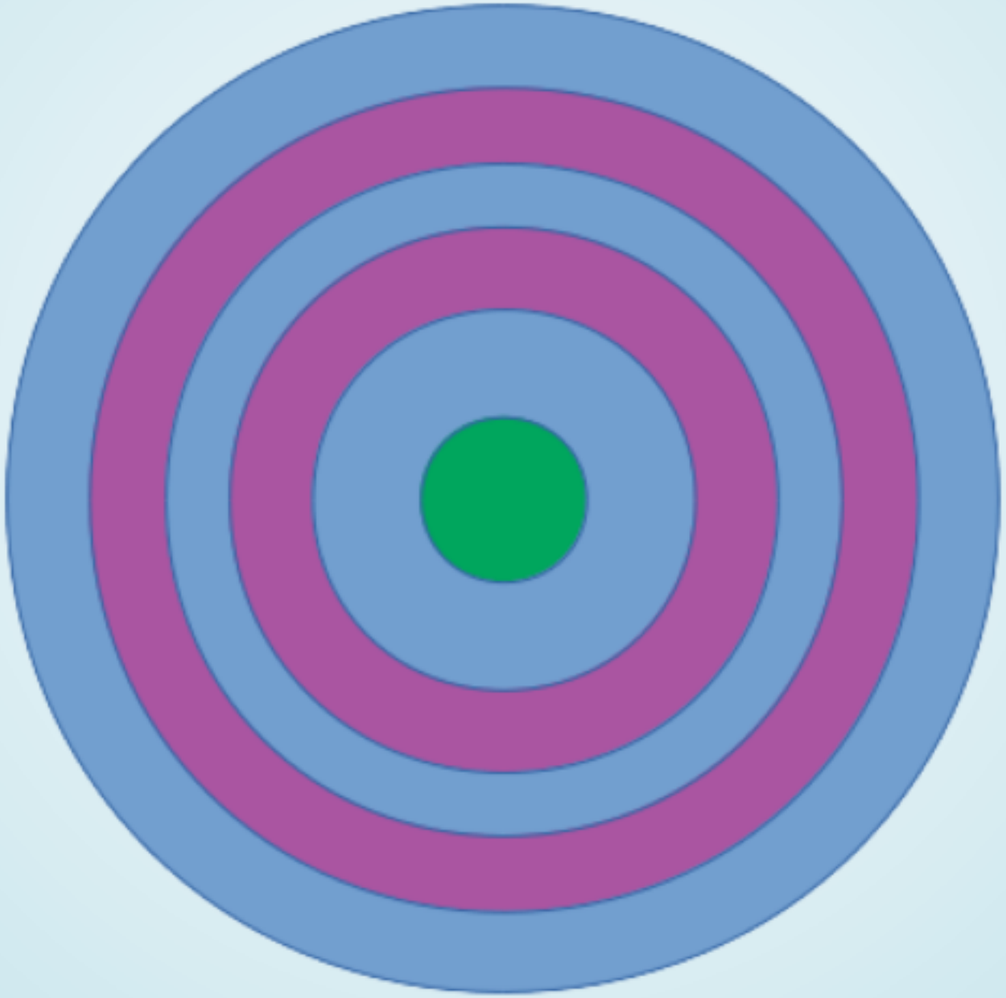
INSIDE OUT



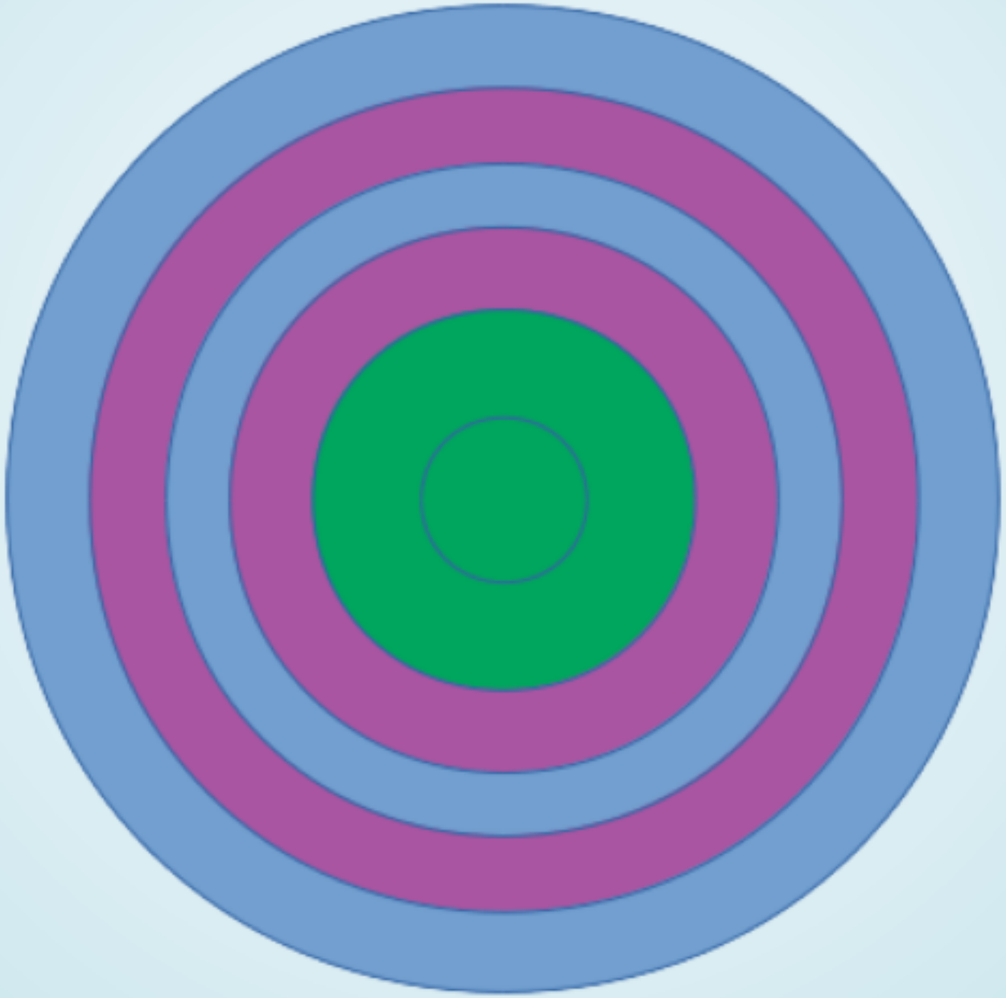
INSIDE OUT


GUI
Inside-Out
Inside-Out
Database / Event Queue
/ External System

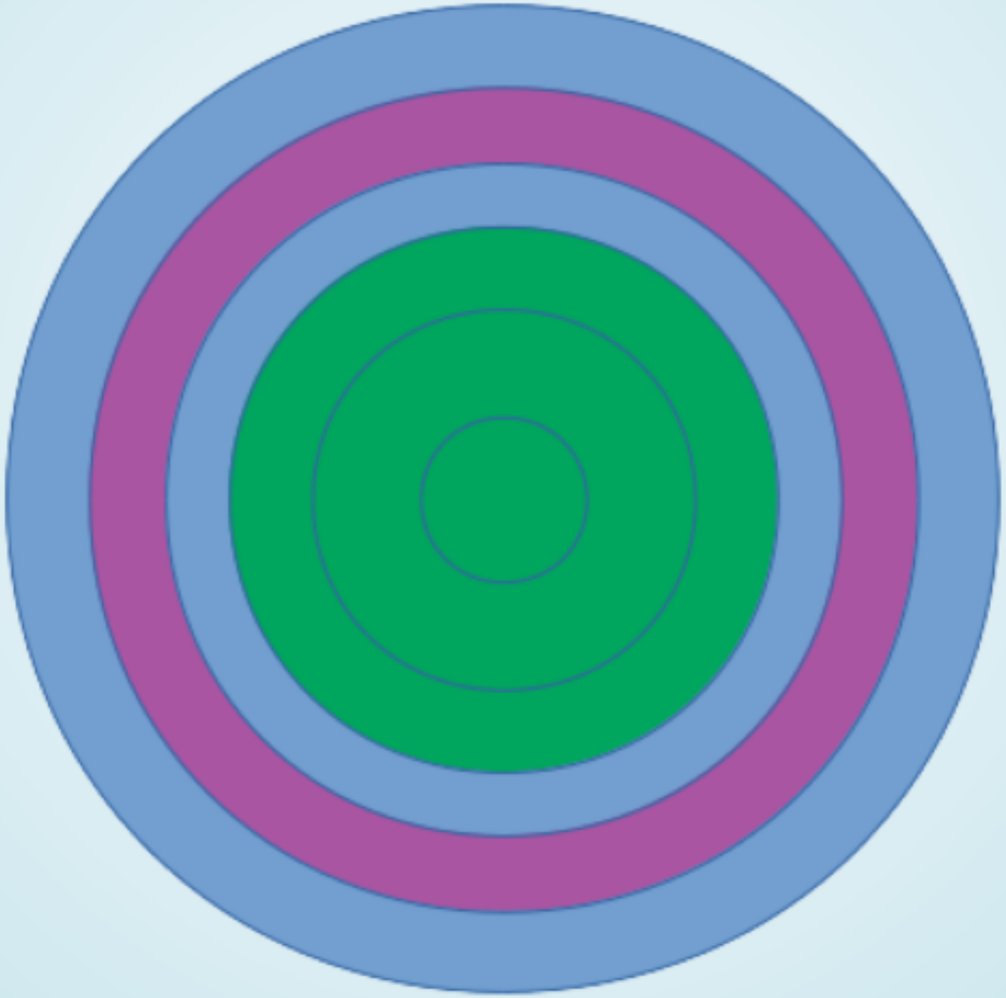
INSIDE OUT



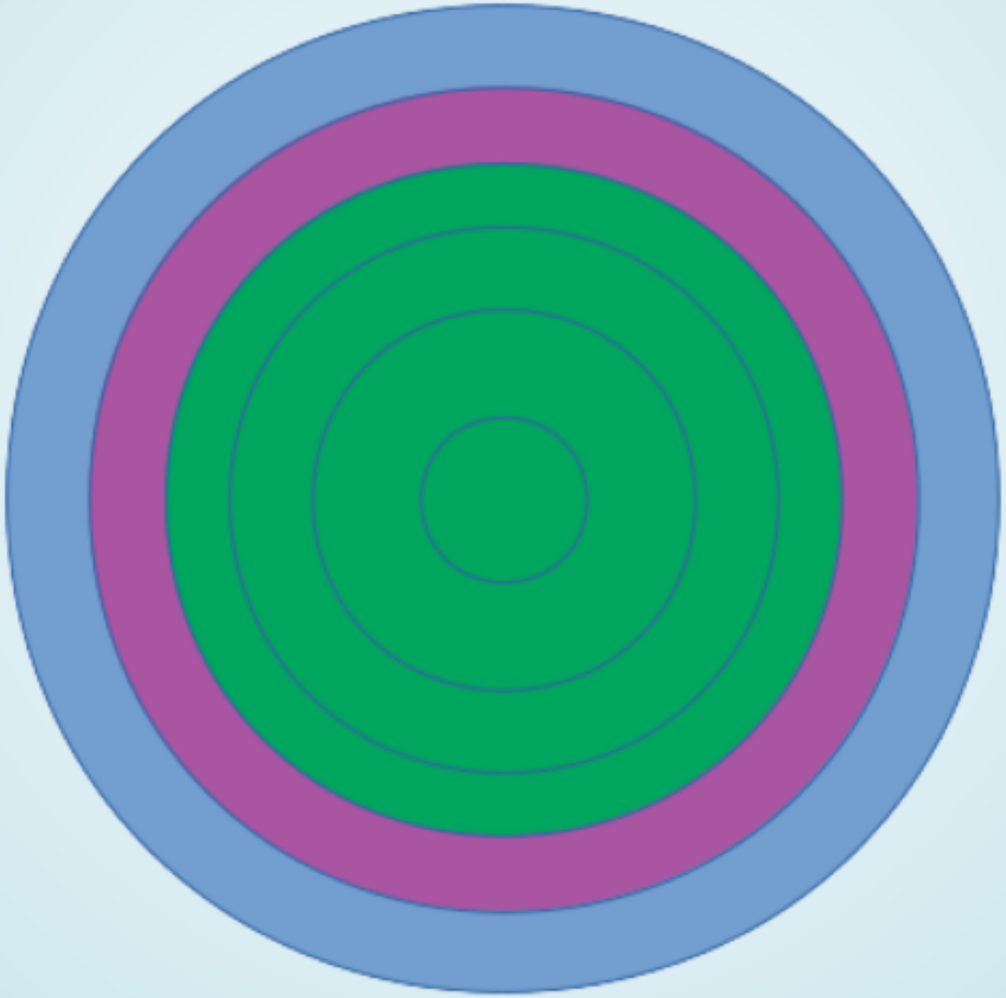
INSIDE OUT



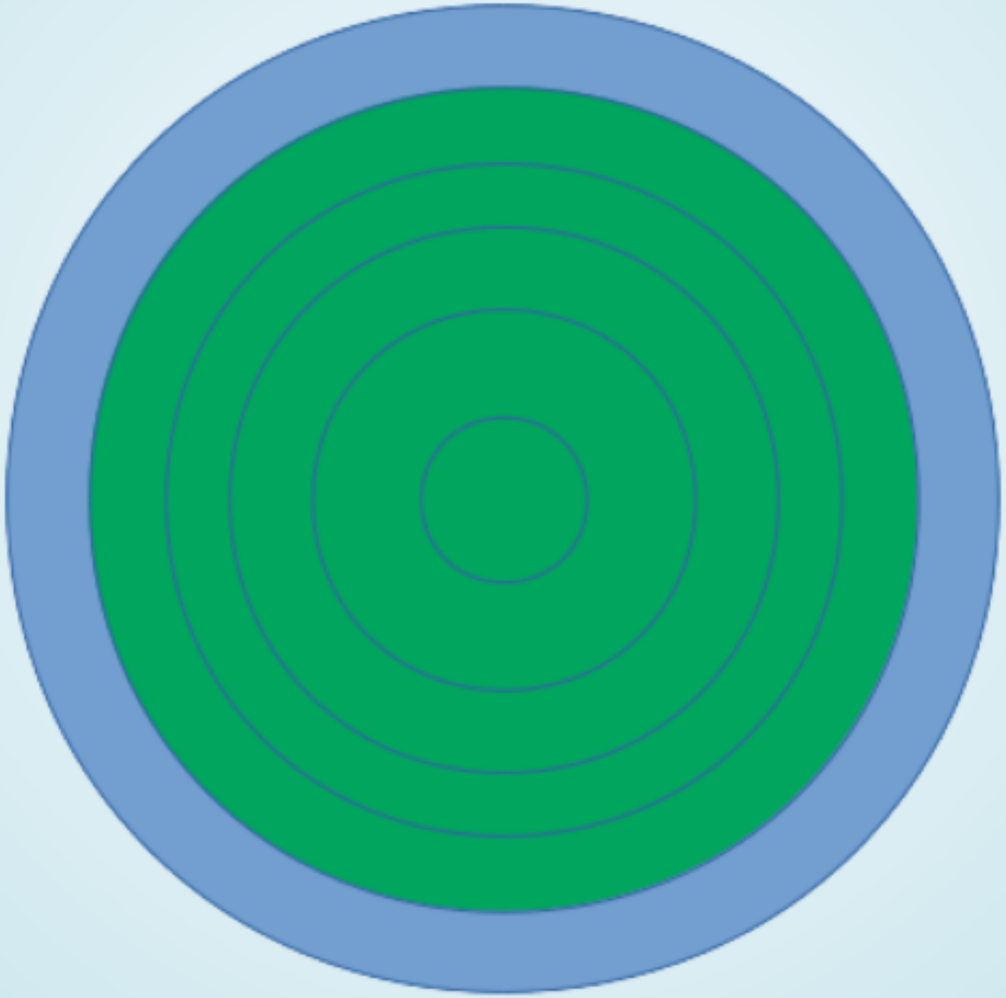
INSIDE OUT



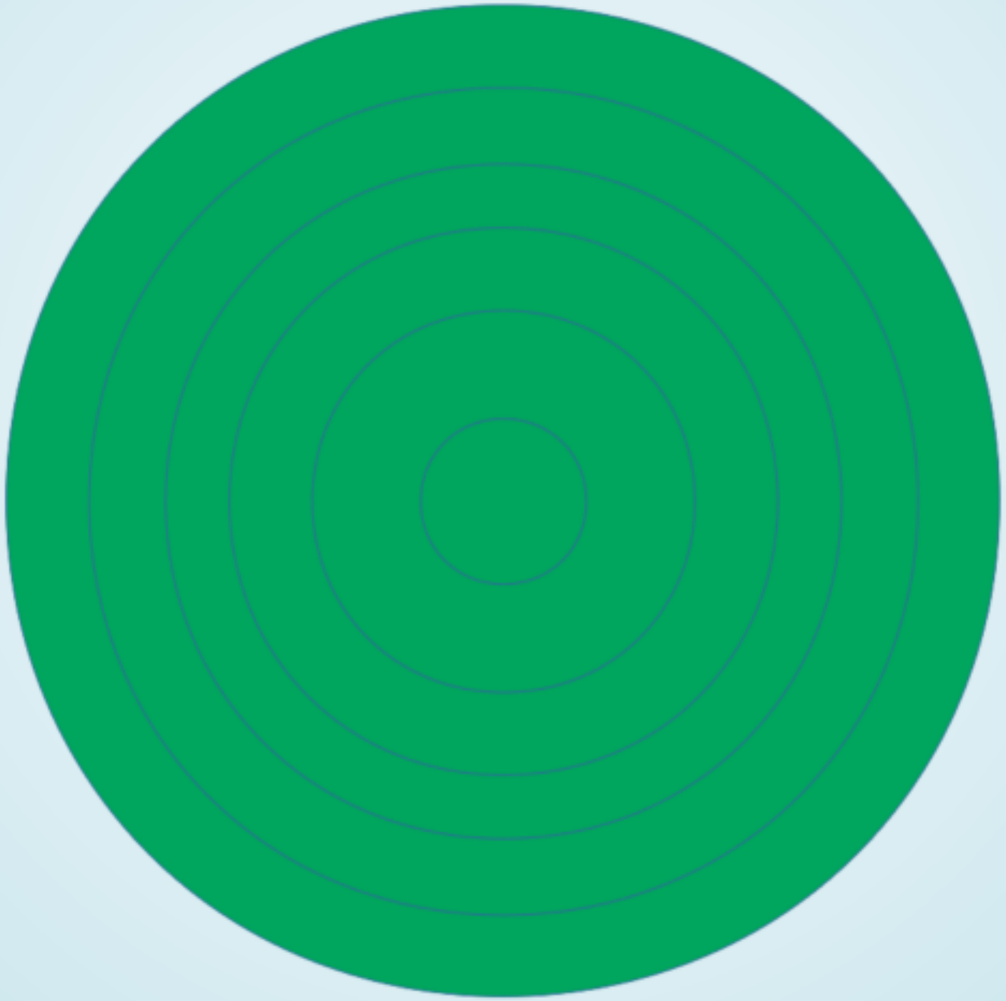
INSIDE OUT



INSIDE OUT

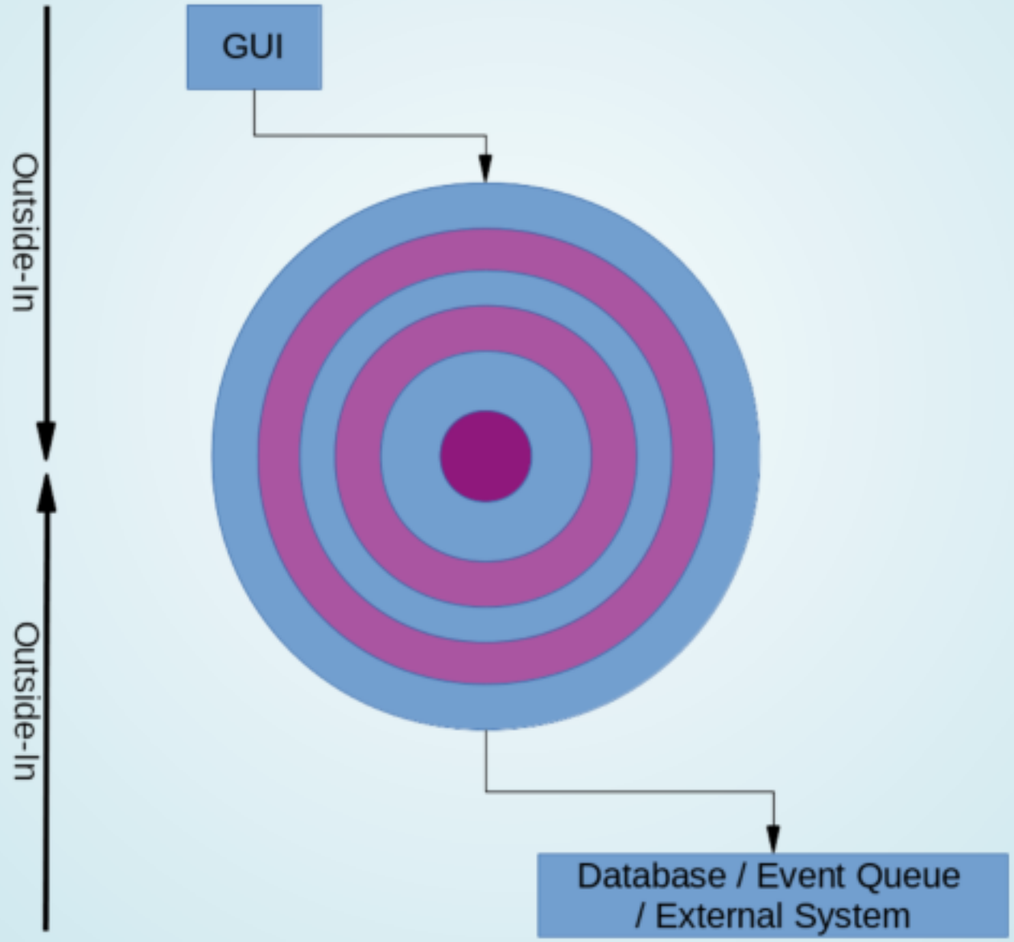


INSIDE OUT

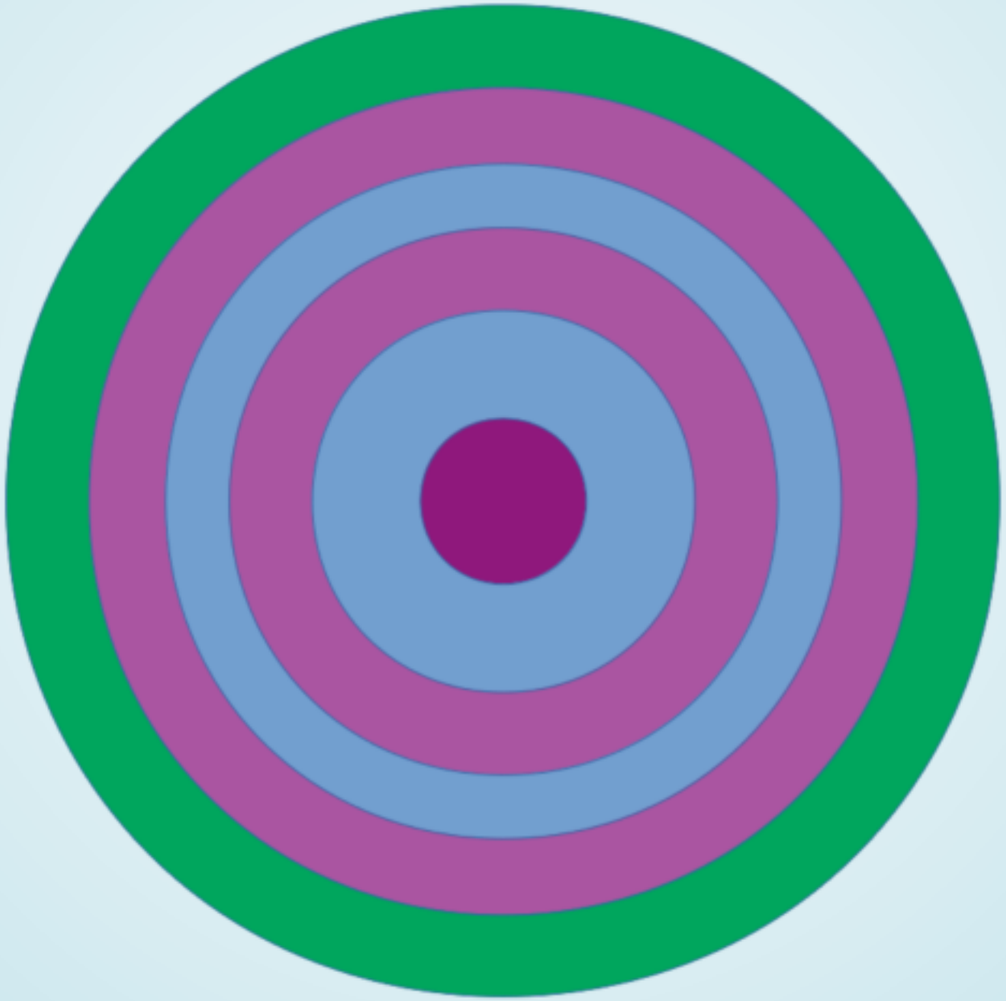


OUTSIDE IN

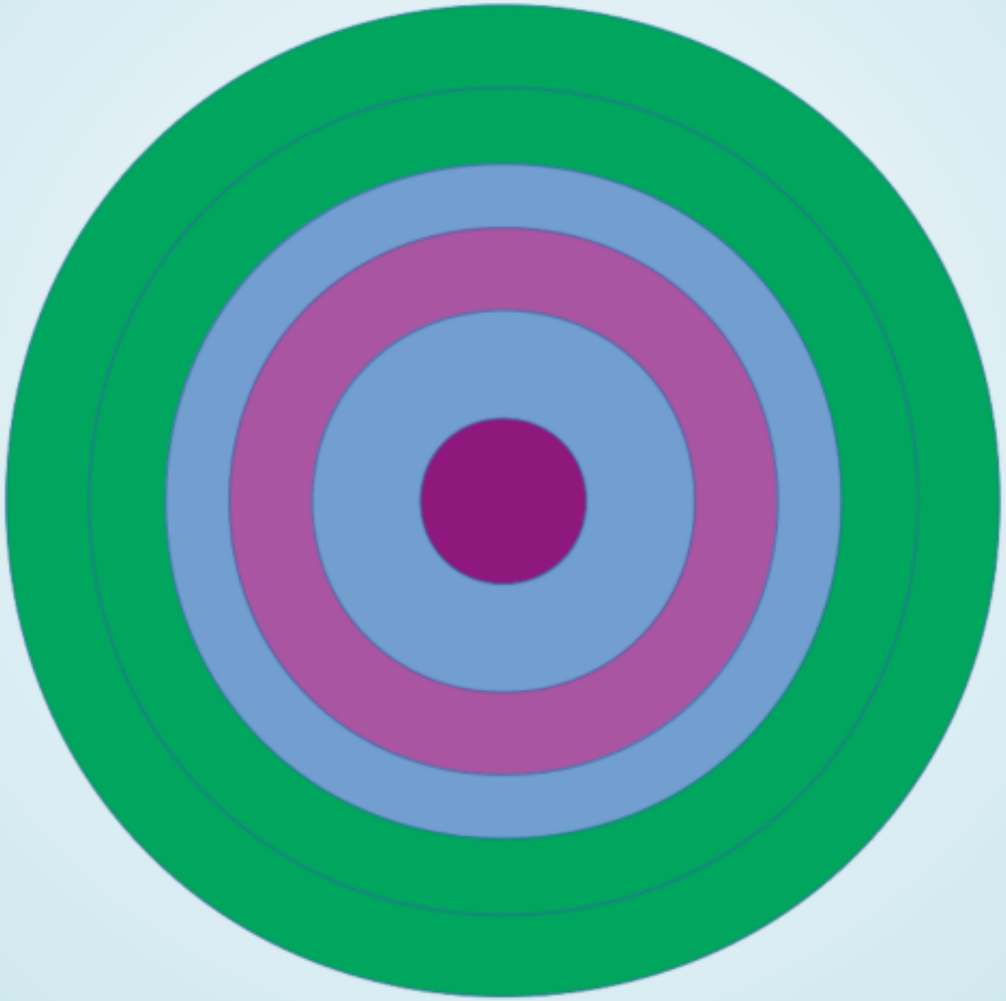
OUTSIDE IN


GUI
Outside-In
Outside-In
Database / Event Queue
/ External System

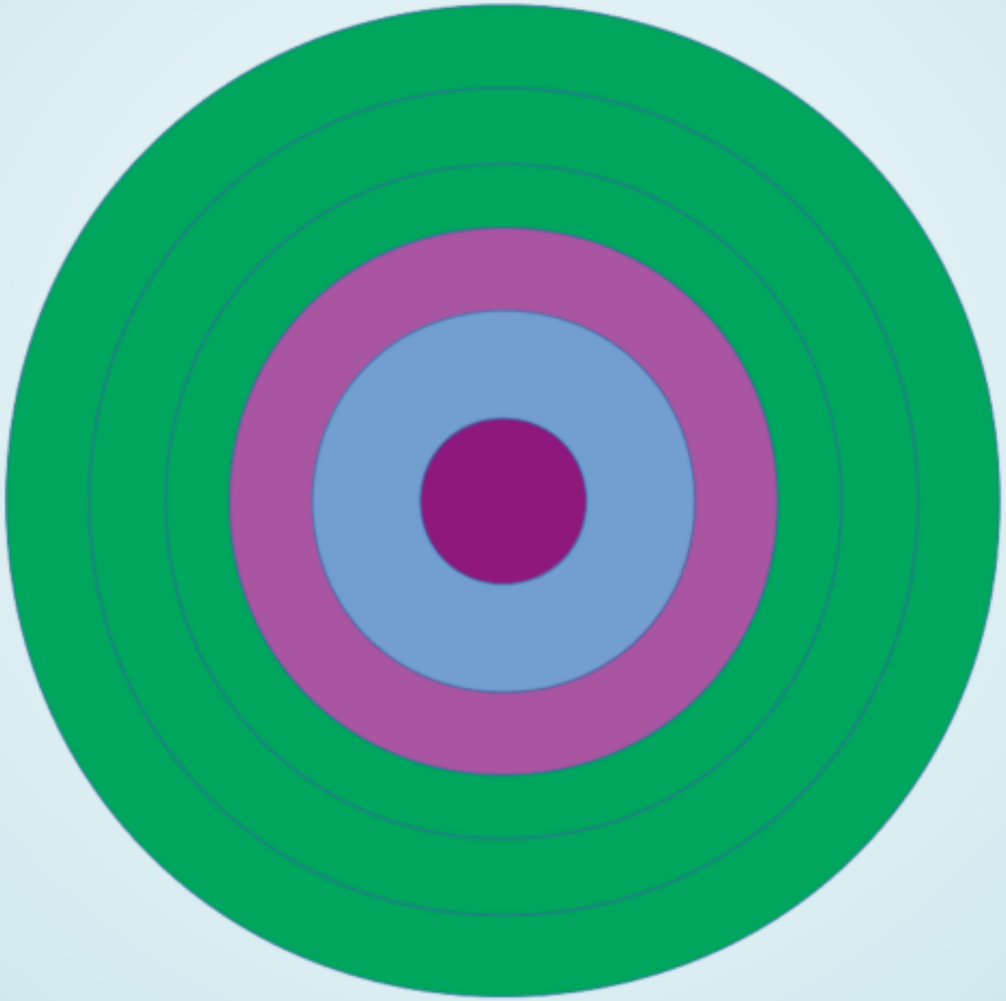
OUTSIDE IN



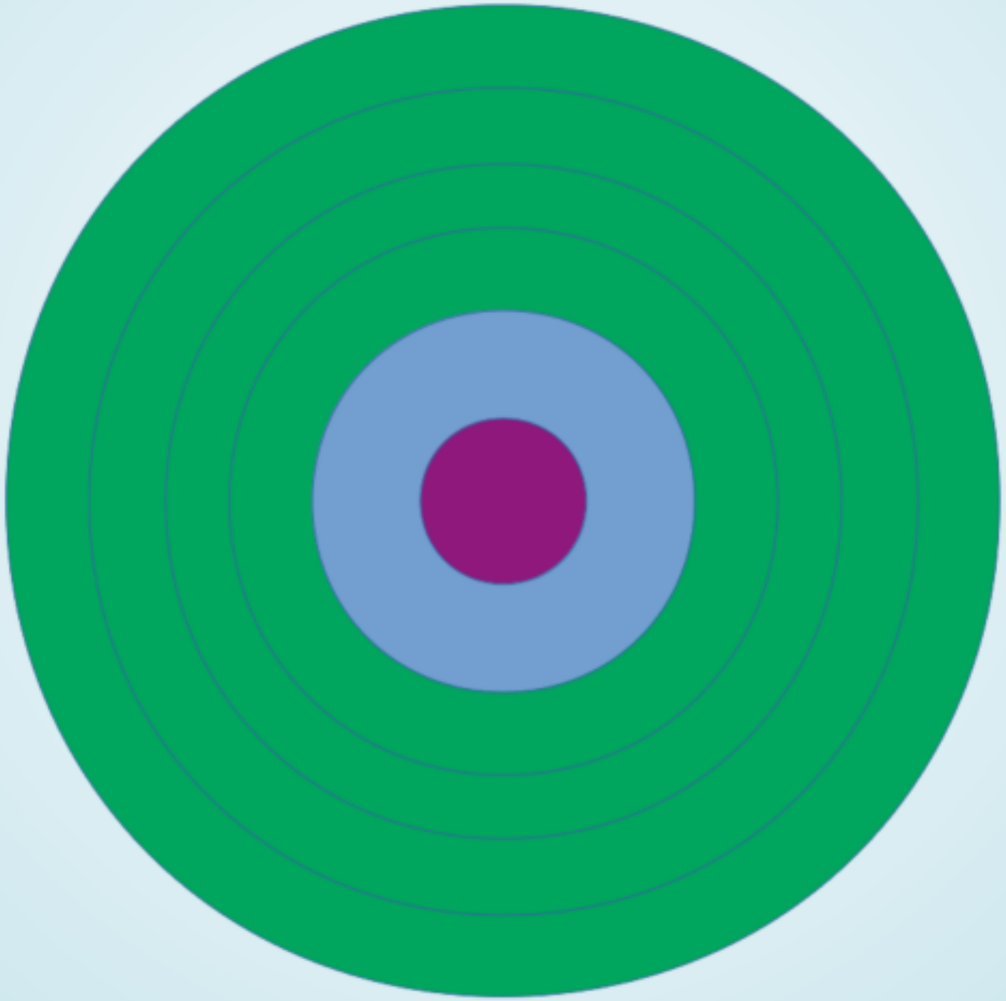
OUTSIDE IN



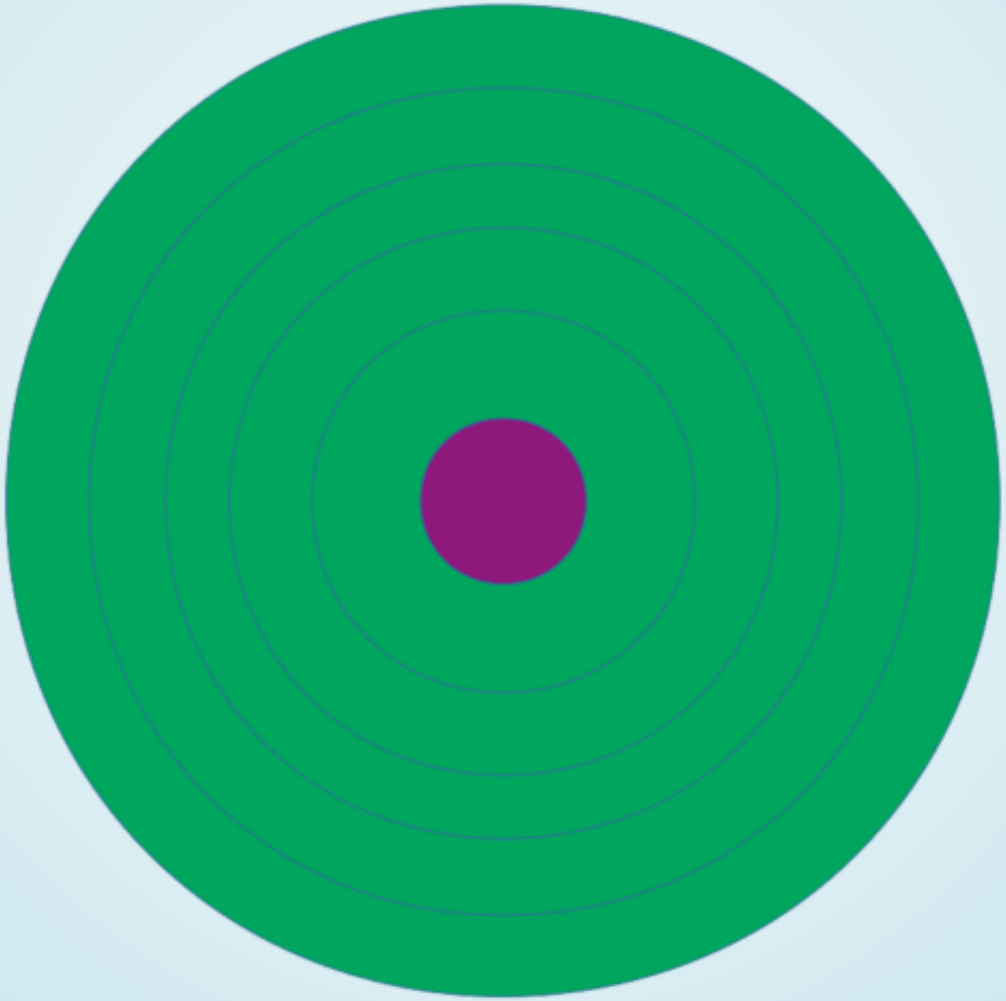
OUTSIDE IN



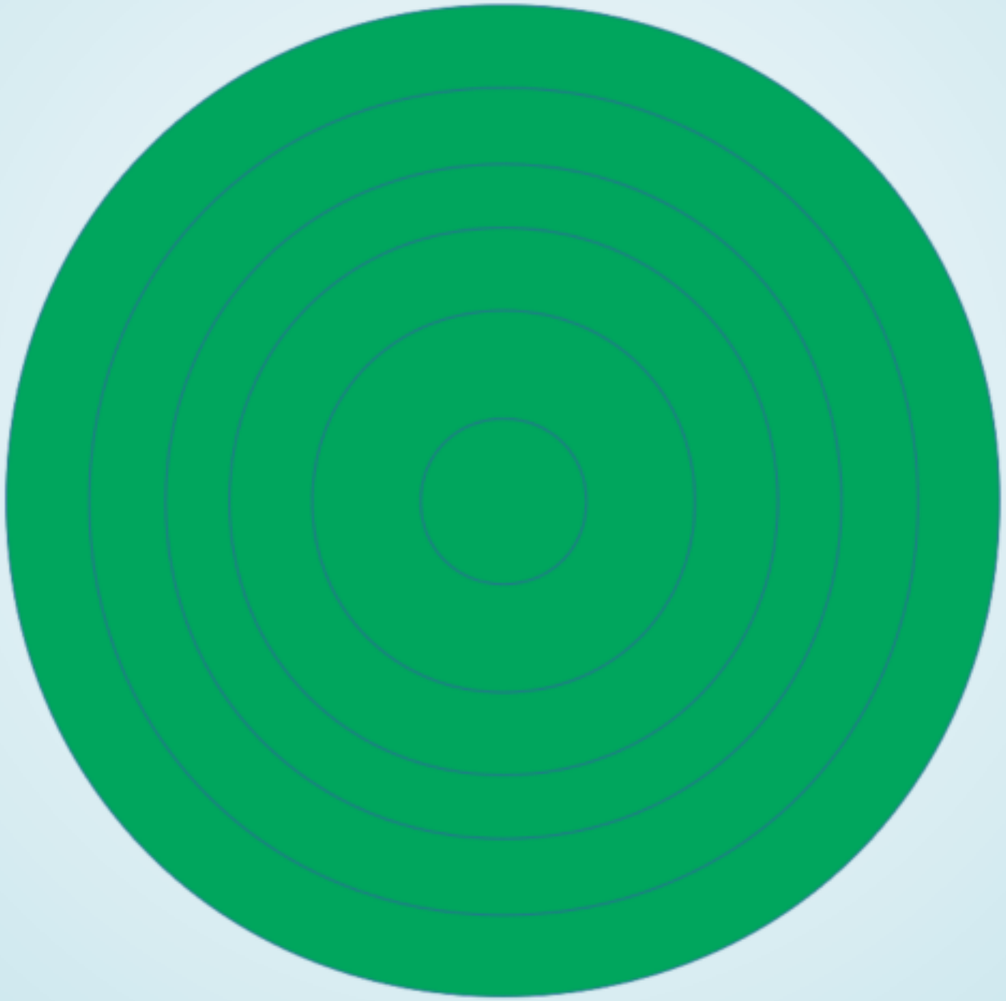
OUTSIDE IN



OUTSIDE IN



OUTSIDE IN




SCHOOLS

COMPARE?

EXAMPLE

BANK KATA


Feiges
FruchtChen
Feige mit Vodka
Likör
20ml e
Kobold

BANK KATA

```
account.deposit(1000);
account.deposit(2000);
account.withdraw(500);
account.printStatement()
```

```
Date        Amount  Balance
14.01.2019  +1000   1000
15.01.2019  +2000   3000
16.01.2019  -500    2500
```

Original by Sandro Mancuso, see Kata-Log by Egga Hartung http://kata-log.rocks/banking-kata

BANK KATA

- Original by Sandro Mancuso
  https://github.com/sandromancuso/Bank-kata
- Kata-Log by Egga Hartung
  (Softwerkskammer Berlin)
  http://kata-log.rocks/banking-kata

Kata-Log

All Topics

Choose Your Level

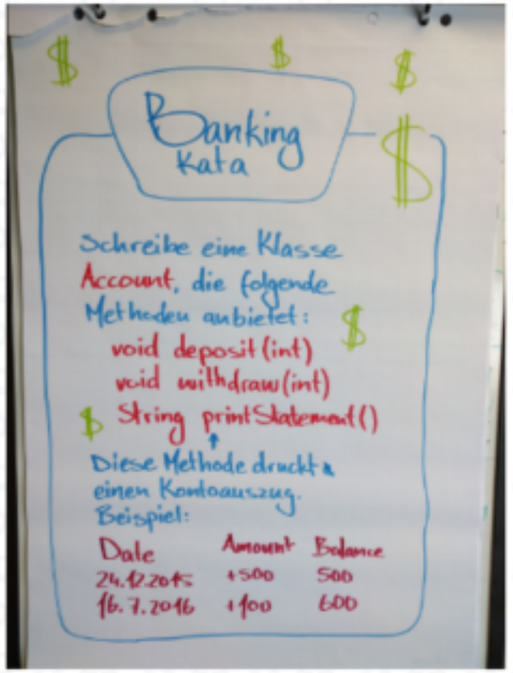
- Starter
- Experienced

Choose a Topic

- Agile
- BDD
- Golden Master
- Mocks
- Outside-In
- Pair-Programming
- Refactoring
- SOLID Principles
- Software-Design
- TDD

Choose a Constraint

Banking Kata



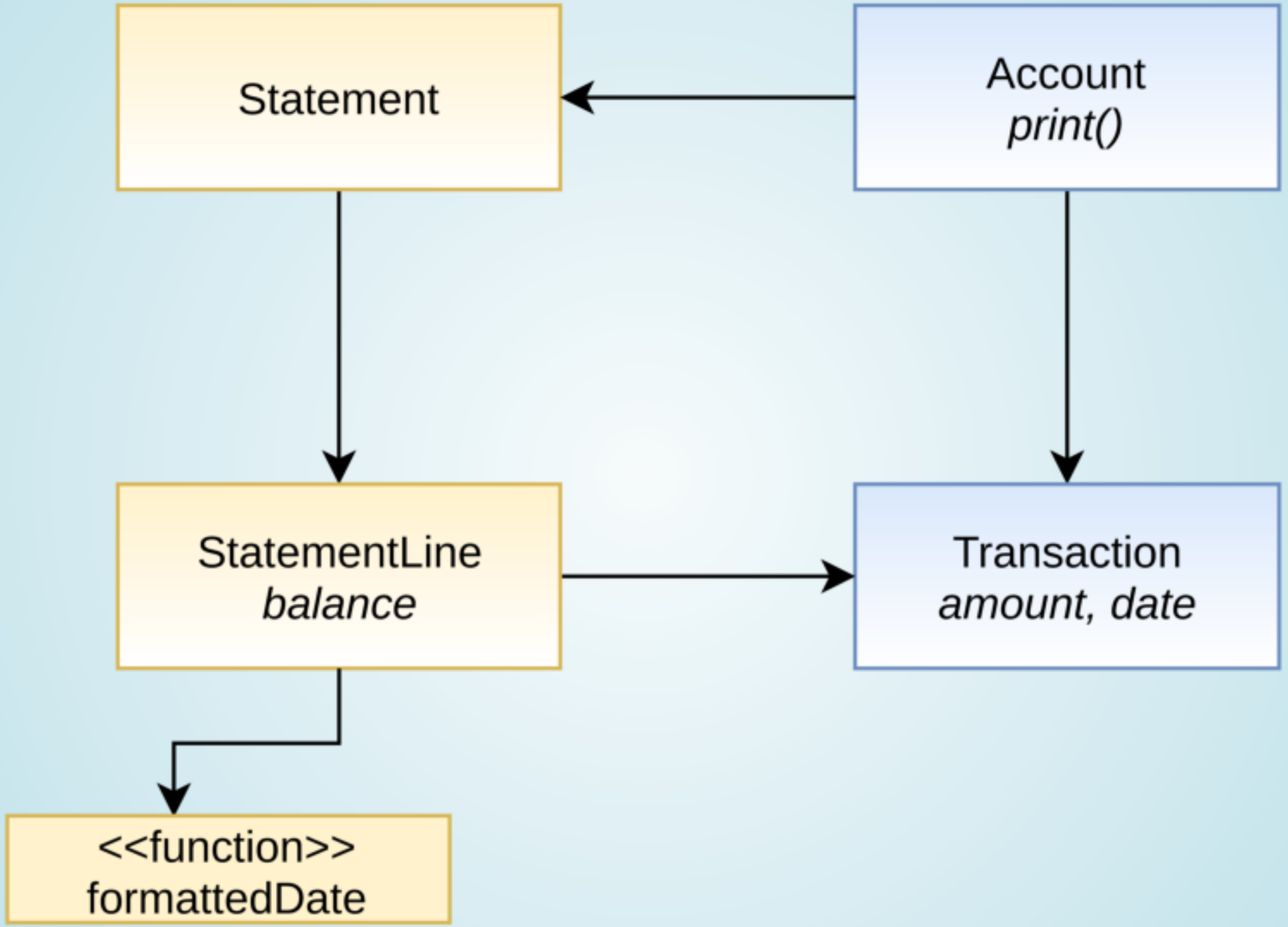
Credits

Inspired by Sandro Mancuso

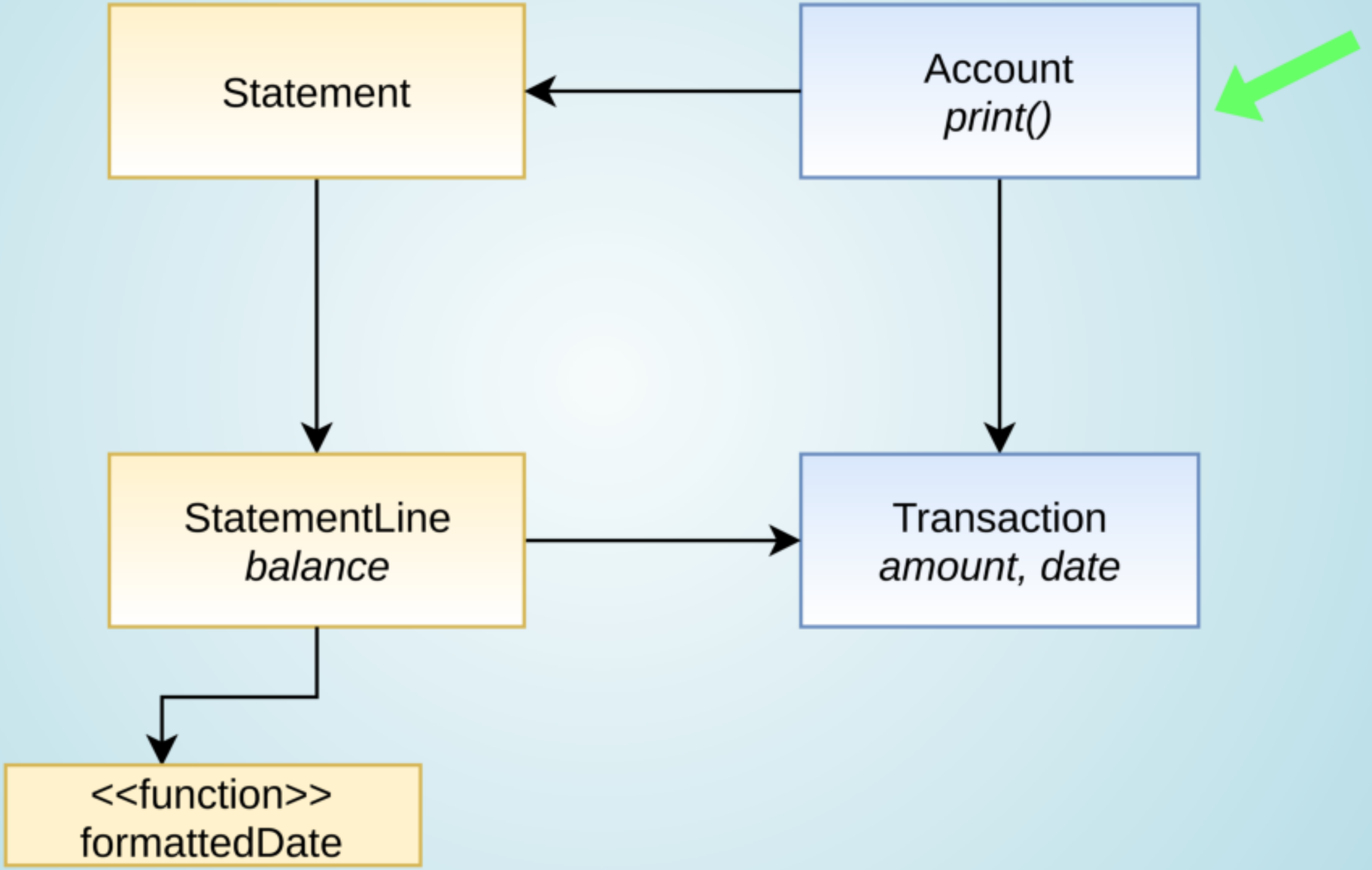
Your Task

Your bank is tired of its mainframe COBOL accounting software and they hired both of you for a greenfield project in - what a happy coincidence

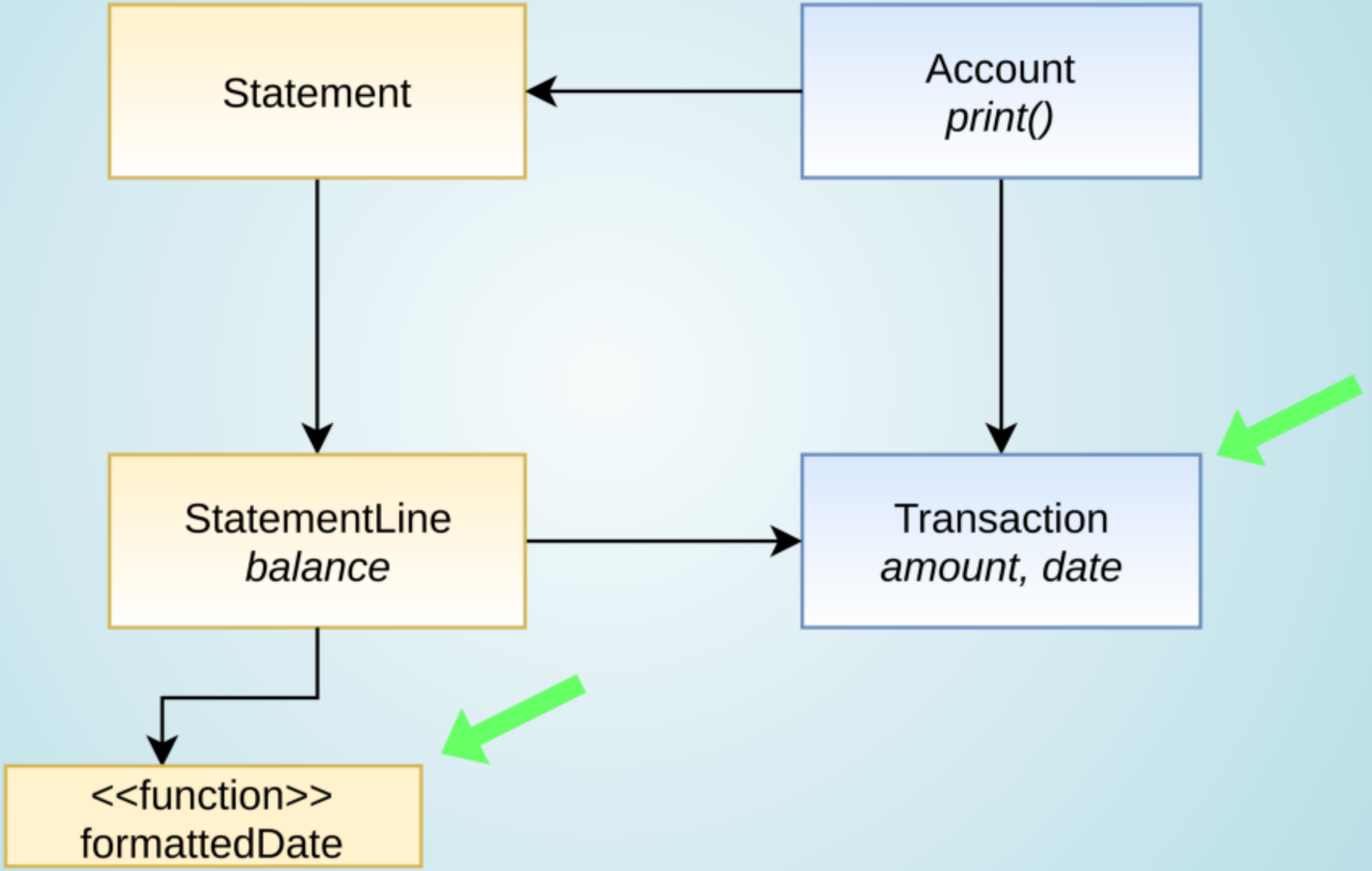
- your favorite programming language!


Statement
Account
print()
StatementLine
balance
Transaction
amount, date
<<function>>
formattedDate

OUTERMOST


Statement
Account
print()
StatementLine
balance
Transaction
amount, date
<<function>>
formattedDate

INNERMOST

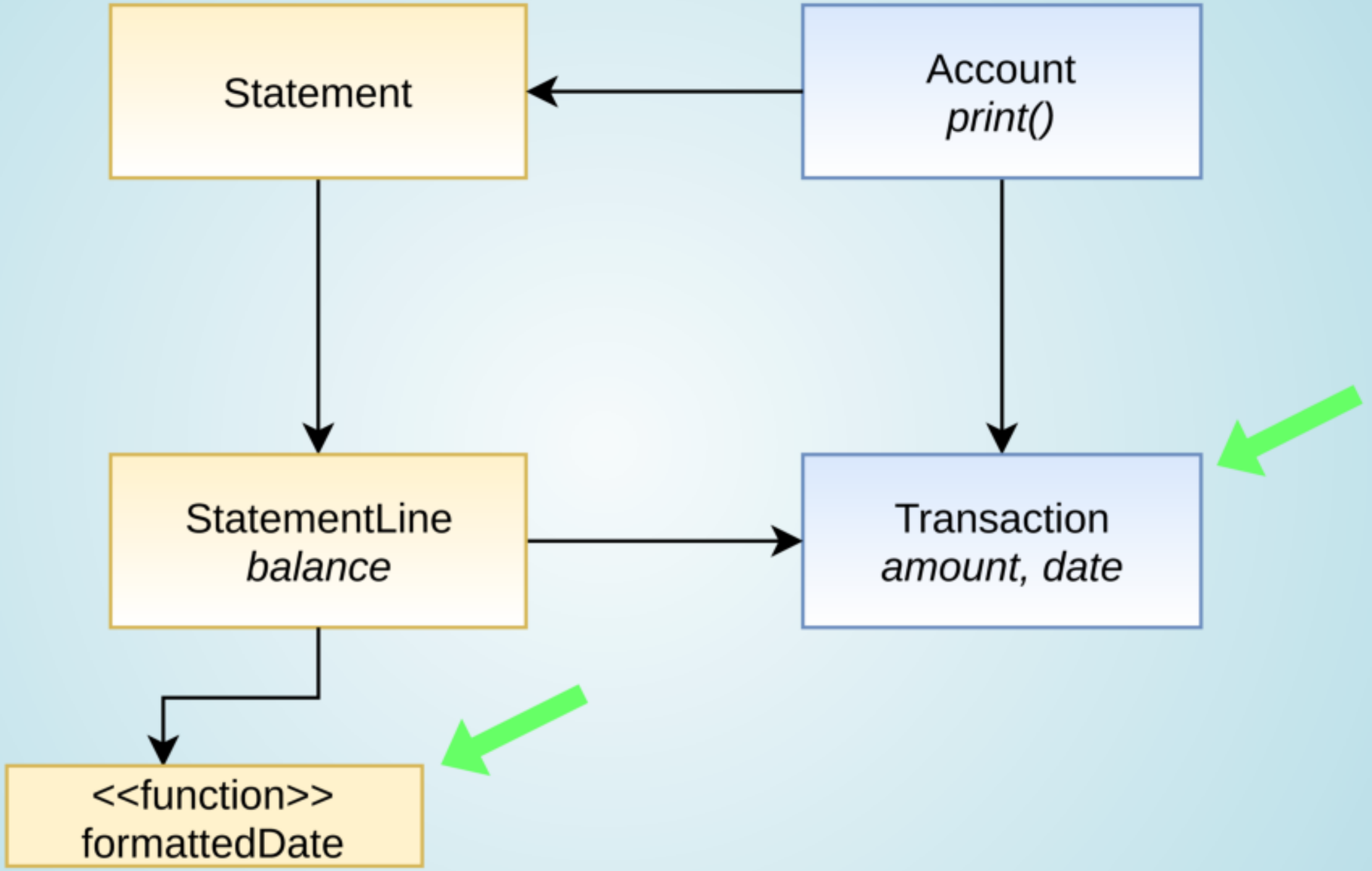

Statement
Account
print()
StatementLine
balance
Transaction
amount, date
<<function>>
formattedDate


CHICAGO
(CLASSIC)

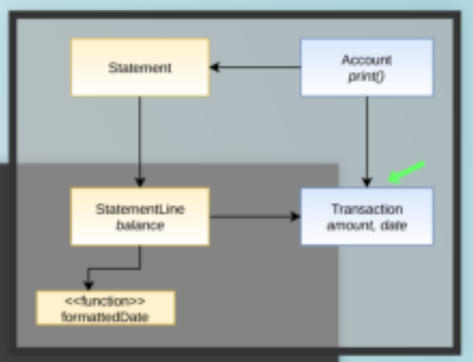
CHICAGO (CLASSIC)

- Inside-Out
- Sociable Tests
- TestDoubles only for External Deps


DISCLAIMER


Statement
Account
print()
StatementLine
balance
Transaction
amount, date
<<function>>
formattedDate

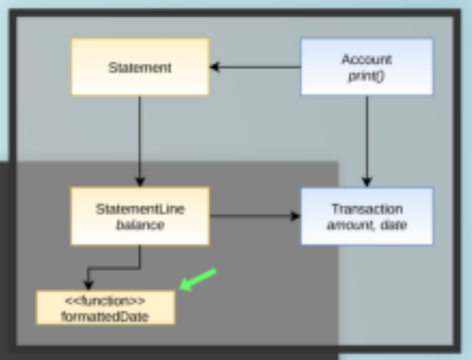
Transaction




```
describe('Transaction', () => {
  it('should hold date & amount', () => {
    const date = new Date(2019, 0, 10);
    const amount = 1200;

    const transaction = new Transaction(date, amount);

    expect(transaction.date).toEqual(date);
    expect(transaction.amount).toEqual(amount);
  })
});
```

```
export class Transaction {
  constructor(date, amount) {
    this.date = date;
    this.amount = amount;
  }
}
```

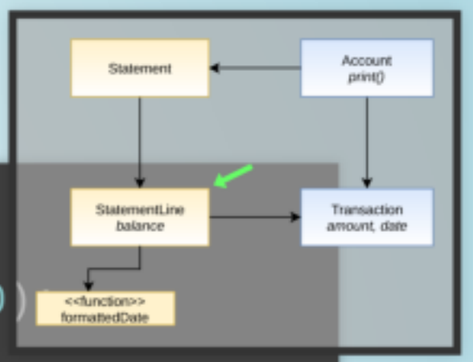
formattedDate




```
describe('formattedDate', () => {
  it('formats a string for a Date', () => {
    expect(
      formattedDate(new Date(2019, 0, 10))
    ).toEqual(
      "10.1.2019"
    );
  });
});
```

```
export const formattedDate = date =>
  `${date.getDate()}` +
  `.${date.getMonth() + 1}` +
  `.${date.getFullYear()}`;
```

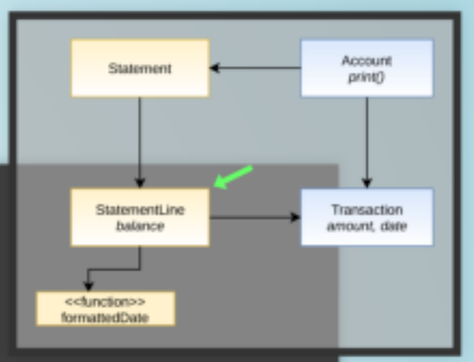
StatementLine




```
describe('StatementLine#toString', () => {
  it('should return a formatted string', () => {
    const transaction = new Transaction(new Date(2019, 0, 10), 1000)

    const statementLine = new StatementLine(transaction, 3000);

    expect( statementLine.toString() ).toEqual(
      "10.1.2019\t+1000\t3000"
    );
  });
});
```

```
export class StatementLine {
  toString() {
    return "10.1.2019\t+1000\t3000";
  }
}
```

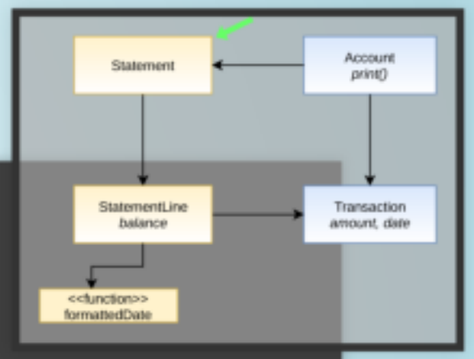
Triangulation




```
    ...
    expect( statementLine.toString() ).toEqual(
      "10.1.2019\t+1000\t3000"
    );
  });

  it('returns a formatted string for 2nd Transaction', () => {
    ...
    expect( statementLine.toString() ).toEqual(
      "14.1.2019\t+500\t2000"
  ...
```

```
import { formattedDate } from "./formattedDate";
export class StatementLine {
...
toString() {
    return `${formattedDate(this.transaction.date)}\t` +
           `+${this.transaction.amount}\t` +
           `${this.balance}`;
  }
}
```

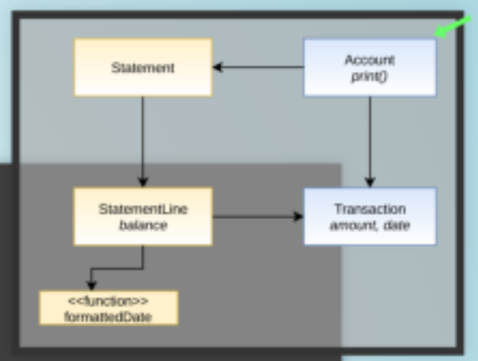
Statement




```
describe('Statement', () => {
  it('should print a blank Statement', () => {...
  it('should print a Statement with 1 Transaction', () => {...
  it('should print a Statement with 2 Transactions', () => {...})
```

```
export class Statement {
  constructor() { ... }
  toString() {
    return "Date\tAmount\tBalance\n" +
  ...
  balanceForLines(lines) {...
  lastLine(lines) {...
  areLinesEmpty(lines) {...
```

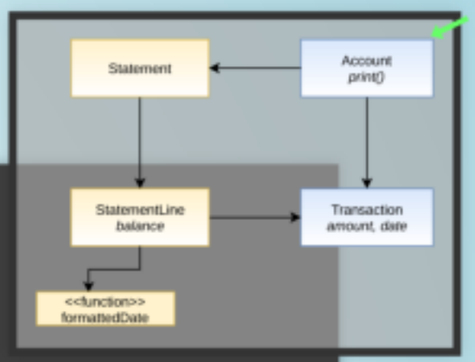
Account




```
  it('should add new Transactions on Deposit', () => {...
  it('should add two new Transactions on two Deposits', () => {
    const account = new Account();

    account.deposit(new Date(2019, 0, 10), 2000);
    account.deposit(new Date(2019, 0, 14), 1000);

    expect( account.transactions ).toEqual( [
      new Transaction(new Date(2019, 0, 10), 2000),
      new Transaction(new Date(2019, 0, 14), 1000),
    ] );
  });
```

```
import { Transaction } from "./Transaction";

export class Account {
  constructor() { this.transactions = []; }

  deposit(date, amount) {
    this.transactions.push(new Transaction(date, amount));
  }
}
```

account.print()




```
describe('Account', () => {
  ...
  it('should print a Statement', () => {
    const printFn = jest.fn();
    const account = new Account();
    account.deposit(new Date(2019, 0, 10), 2000);
    account.deposit(new Date(2019, 0, 14), 1000);
    account.print(printFn);
    expect(printFn).toHaveBeenCalledWith((new Statement(account)).toString());
  });
});
```

```
import { Transaction } from "./Transaction";
import { Statement } from "./Statement";

export class Account {
  ...
  print(printFn) {
    const statement = new Statement(this);
    printFn(statement.toString());
  }
```

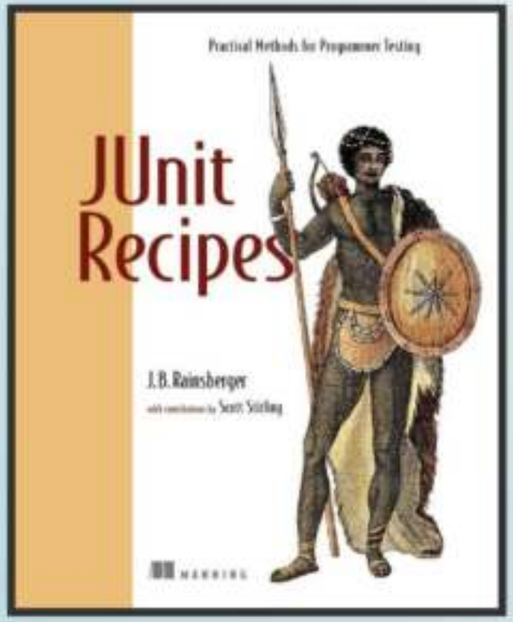
CONSEQUENCES OF CHICAGO CLASSIC

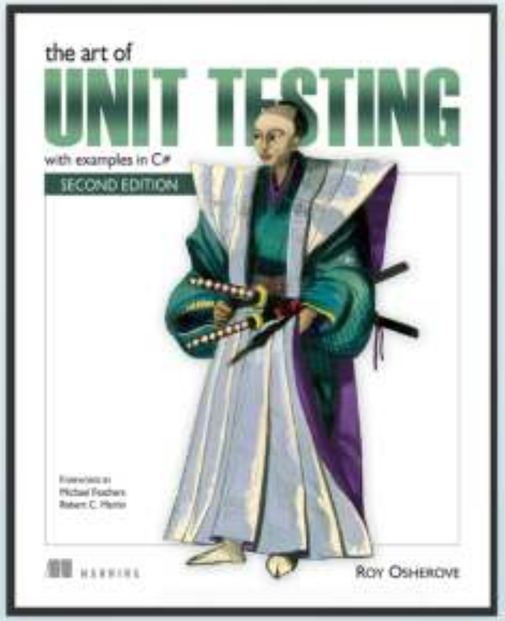
- - Can be fragile
- - Can be slow
- - Defect Localization is hard

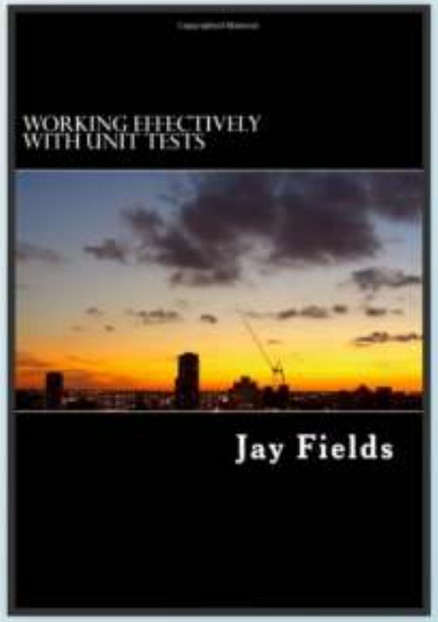

CHICAGO
(MODERN)

CHICAGO (MODERN)

- J.B. Rainsberger
- Roy Osherove
- Jay Fields
- ...



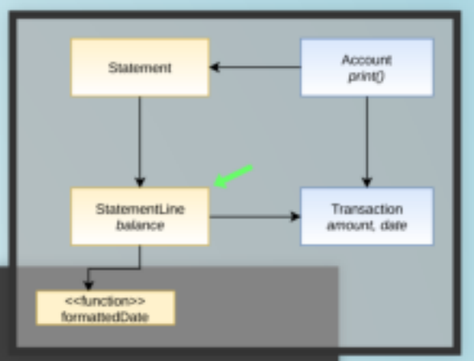




CHICAGO (MODERN)

- Inside-Out
- Solitary Tests
- Lot of Stubs / Some Mocks

SUT: StatementLine
NOT SUT: Transaction, formattedDate




```
describe('StatementLine#toString', () => {
  it('should return a formatted string', () => {
    const transaction = new Transaction(new Date(2019, 0, 10), 1000);
    const statementLine = new StatementLine(transaction, 3000);
    expect( statementLine.toString() ).toEqual(
      "10.1.2019\t+1000\t3000"
    );
  });
```

```
import { formattedDate } from "./formattedDate";

export class StatementLine {
  constructor(transaction, balance) {
    this.transaction = transaction;
    this.balance = balance;
  }
  toString() {
    return `${formattedDate(this.transaction.date)}\t+${this.transaction.amount}
  }
}
```

SUT: StatementLine

```
it('should return a formatted string', () => {
  const transaction = new Transaction(new Date(2019, 0, 10), 1000);
  const statementLine = new StatementLine(transaction, 3000);
  ...
});
```

↓ Refactoring: replace Class with Stub

```
it('should return a formatted string', () => {
  const transaction = {date: new Date(2019, 0, 10), amount: 1000};
  const statementLine = new StatementLine(transaction, 3000);
  ...
});
```

SUT: StatementLine

```
export class StatementLine {
  ...
  toString() {
    return `${formattedDate(this.transaction.date)}\t+${this.transaction.amount}
  }
}
```

↓ Refactoring: Inject Function Dependency

```
export class StatementLine {
  ...
  toString(dateFormat) {
    return `${dateFormat(this.transaction.date)}\t+${this.transaction.amount}\t$
  }
}
```

```
it('should return a formatted string', () => {
  ...
  expect( statementLine.toString() ).toEqual(
    "10.1.2019\t+1000\t3000"
});
```

↓ Refactoring: Inject Function Dependency

```
it('should return a formatted string', () => {
  ...
  const germanDateFormat = jest.fn().mockReturnValue("10.1.2019");
  ...
  expect( statementLine.toString(germanDateFormat) ).toEqual(
    "10.1.2019\t+1000\t3000"
  );
});
```

Refactoring: Rename Function

formattedDate → germanDateFormat

```
import { germanDateFormat } from "./germanDateFormat";

describe('germanDateFormat', () => {
  it('should return a string for a Date formatted to the German standard', () =>
    expect(germanDateFormat(new Date(2019, 0, 10))).toEqual("10.1.2019");
  });
});
```

```
export const germanDateFormat = date =>
  `${date.getDate()}.${date.getMonth() + 1}.${date.getFullYear()}`;
```

CONSEQUENCES OF CHICAGO MODERN

- + ~~can be fragile~~ mostly stable
- + ~~Can be slow~~ blazing fast
- + Defect Localization is ~~hard~~ easy
- - Additional Integration Tests needed

GENERAL CHICAGO CONSEQUENCES

- + Useful for Emergent Design/Experimentation
- + Applicable in most situations
- + Easy to learn
- - Needs more Refactoring than London
- - Dead/useless code possible: YAGNI


LONDON

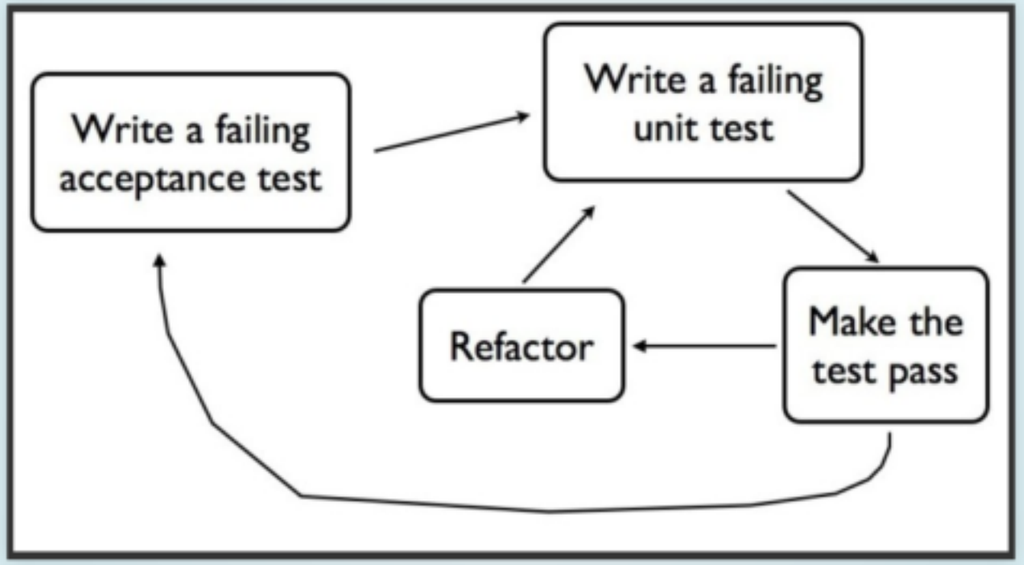
LONDON

=

OUTSIDE-IN WITH MOCKS

LONDON

Double Loop ATDD

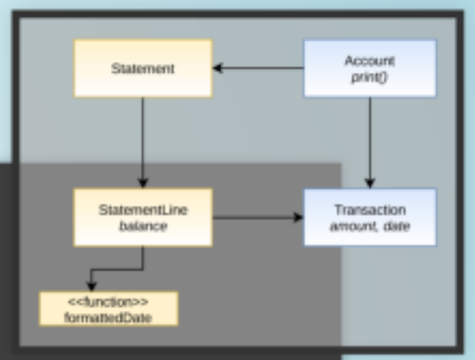


stolen from Emily Bache

LONDON

- Double Loop ATDD
- Outside-In
- Mocks as main Verification Mechanism
- Less Refactoring

Acceptance Spec




```
describe('Banking Kata Acceptance Spec', () => {
  it('should print a Statement with two Deposits', () => {
    const printerFn = jest.fn();
    const account = new Account();
    account.deposit(Date(2012, 0, 10), 1000);
    account.deposit(Date(2012, 0, 13), 2000);

    account.print(printerFn);

    expect(printerFn).toHaveBeenCalledWith(
      "Date\tAmount\tBalance\n" +
      "10.1.2012\t+1000\t1000\n" +
      "13.1.2012\t+2000\t3000\n"
    );
  });
});
```

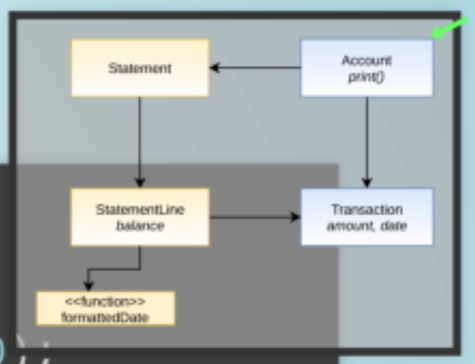
Acceptance Spec

```
describe('Banking Kata Acceptance Spec', () => {
  it('should print a Statement with two Deposits', () => {
  ...
```

X-out

```
xdescribe('Banking Kata Acceptance Spec', () => {
  it('should print a Statement with two Deposits', () => {
  ...
```

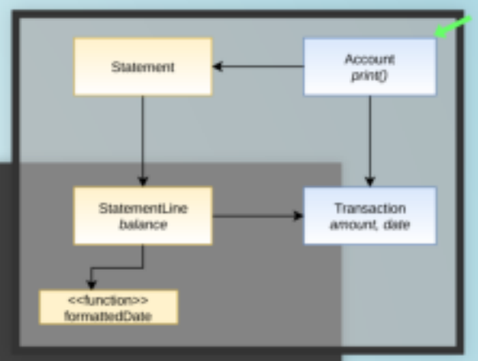
Account Spec: deposit




```
describe('Account', () => {
  it('should add StatementLine of deposit to Statement', () => {
    const statement = {addLine: jest.fn()};
    const transaction = new Transaction(new Date(2012, 0, 13), 1000);
    const account = new Account(statement);

    account.deposit(new Date(2012, 0, 13), 1000);
    expect(statement.addLine).toHaveBeenCalledWith(
      transaction, 1000
    )
  });
```

```
class Account {
  constructor(statement) {
    this.statement = statement;
  }
  deposit(date, value) {
    this.statement.addLine(new Transaction(date, value), value);
  }
}
```

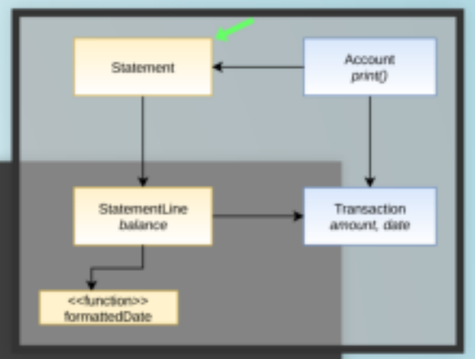
Account Spec: printStatement




```
describe('Account', () => {
  it('should add StatementLine of deposit to Statement', () => {
  ...

  it('should print Statement', () => {
    const statement = {print: jest.fn()};
    const account = new Account(statement);
    const printFn = jest.fn();
    account.print(printFn);
    expect(statement.print).toHaveBeenCalledWith(printFn);
  });
});
```

```
class Account {
  ...
  printStatement(fn) {
    this.statement.print(fn);
  }
}
```

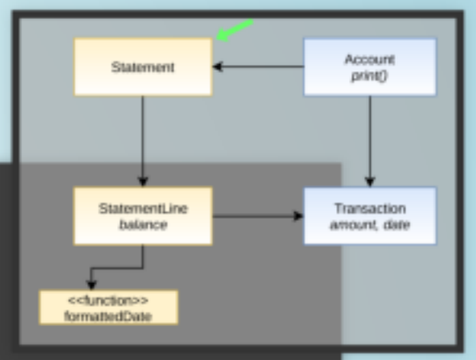
Statement




```
describe('Statement', () => {
  it('should print Header', () => {
    const printFn = jest.fn();
    const statement = new Statement()
    statement.print(printFn);
    expect(printFn).toHaveBeenCalledWith('Date\tAmount\tBalance');
  });
});
```

```
class Statement {
  print(printFn) {
    printFn('Date\tAmount\tBalance');
  }
}
```

Statement



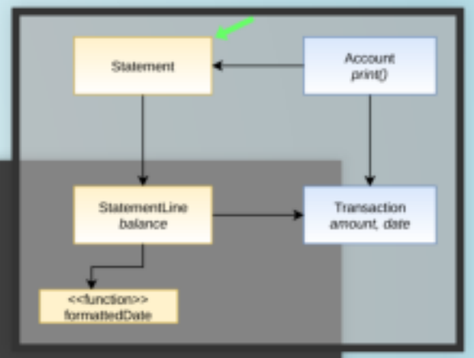

```
class Statement {
  print(printFn) {
    printFn('Date\tAmount\tBalance');
  }
}
```

Refactoring: Extract Constant

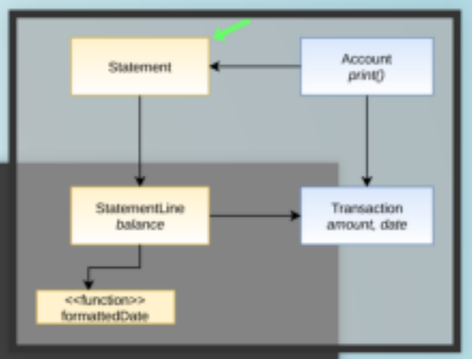
```
class Statement {
  constructor() {
    this.STATEMENT_HEADER = 'Date\tAmount\tBalance';
  }

  print(printFn) {
    printFn(this.STATEMENT_HEADER);
  }
}
```

Statement Spec

```
describe('Statement', () => {
  it('should print Header', () => {
    const printFn = jest.fn();
    const statement = new Statement()
    statement.print(printFn);
    expect(printFn).toHaveBeenCalledWith(statement.STATEMENT_HEADER);
  });
});
```

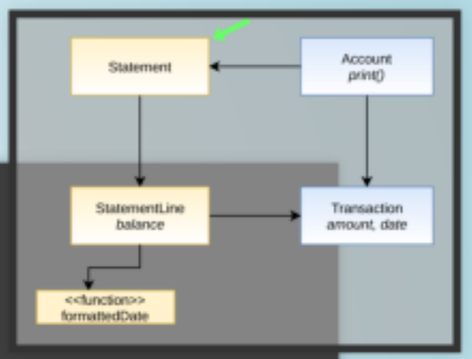


Statement Spec




```
...
  it('should print each line', () => {
    const printFn = jest.fn();
    const statementLine1 = {print: jest.fn()};
    const statementLine2 = {print: jest.fn()};

    const statement = new Statement()
    statement.addLine(statementLine1);
    statement.addLine(statementLine2);
    statement.print(printFn);

    expect(statementLine1.print).toHaveBeenCalledWith(printFn);
    expect(statementLine2.print).toHaveBeenCalledWith(printFn);
  });
});
```

Statement




```
class Statement {
  constructor() {
    this.STATEMENT_HEADER = 'Date\tAmount\tBalance';
    this.lines = [];
  }

  addLine(statementLine) {
    this.lines.push(statementLine);
  }

  print(printFn) {
    printFn(this.STATEMENT_HEADER);
    this.lines.forEach(l => l.print(printFn));
  }
}
```

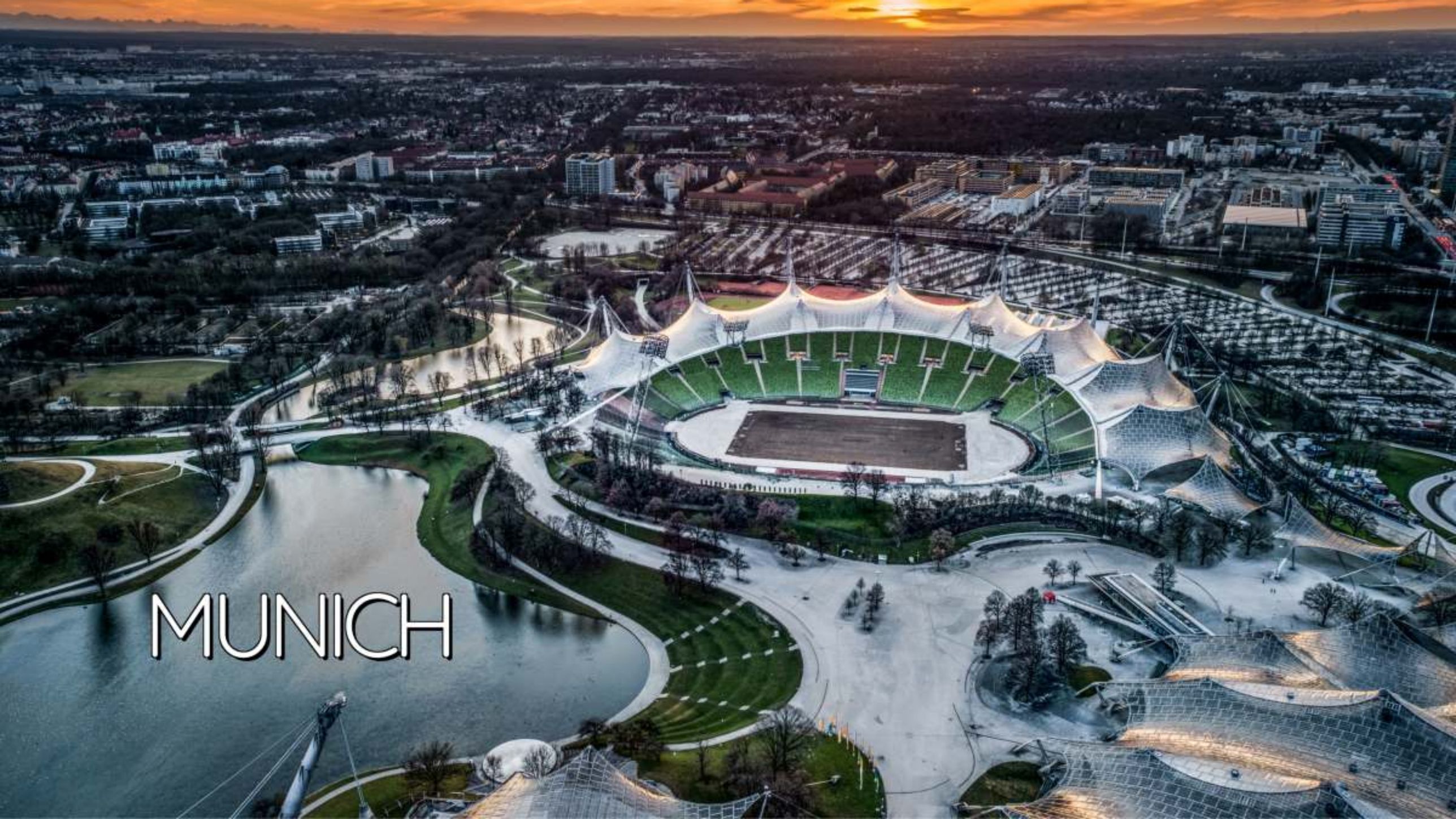
LONDON CONSEQUENCES

- + No YAGNI
- + Obvious next step
- Leads to "Tell Don't Ask"-design
- Easier for known designs
- - Design harder to refactor
- - Mocking is hard


TEST DOUBLES
MOCKS


156


BCCL


MUNICH

MUNICH

=

FAKE-IT OUTSIDE-IN

MUNICH

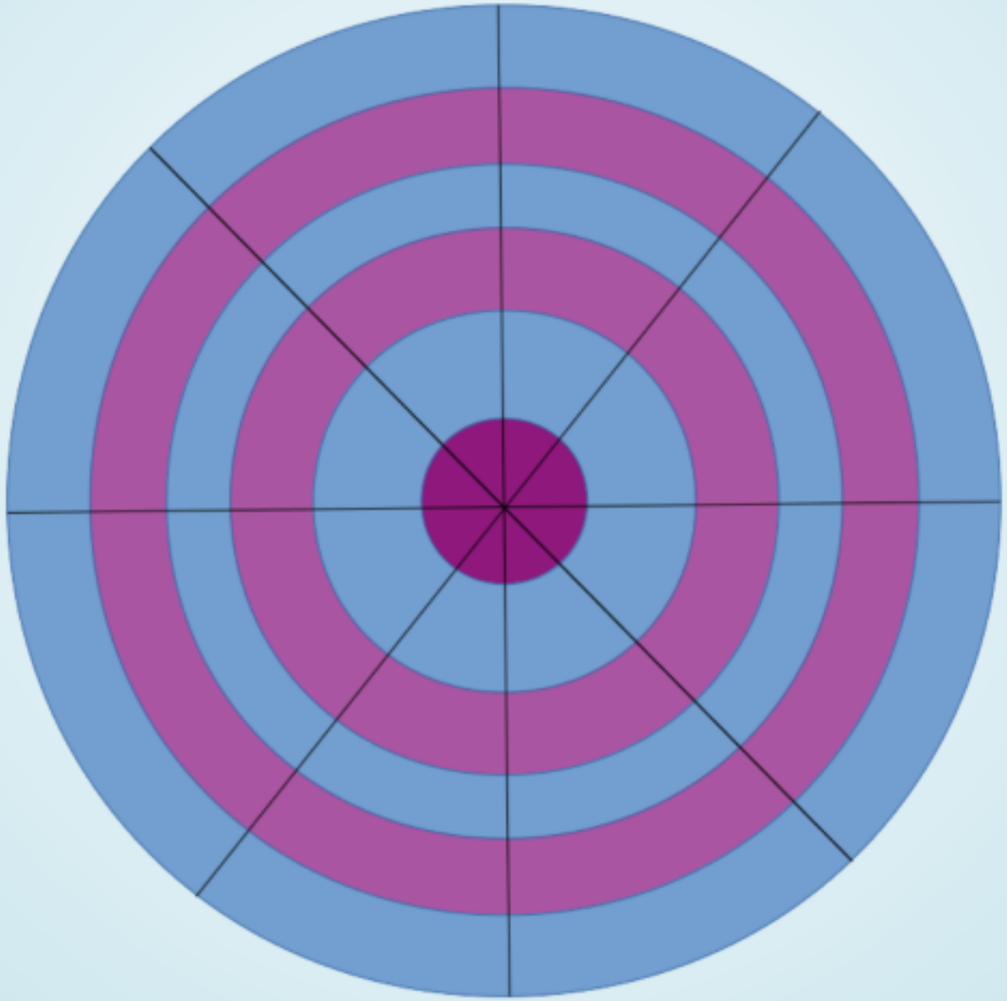
- Outside-In
- with or without Mocks
- Very Large Amount of Refactorings

INFLUENCES

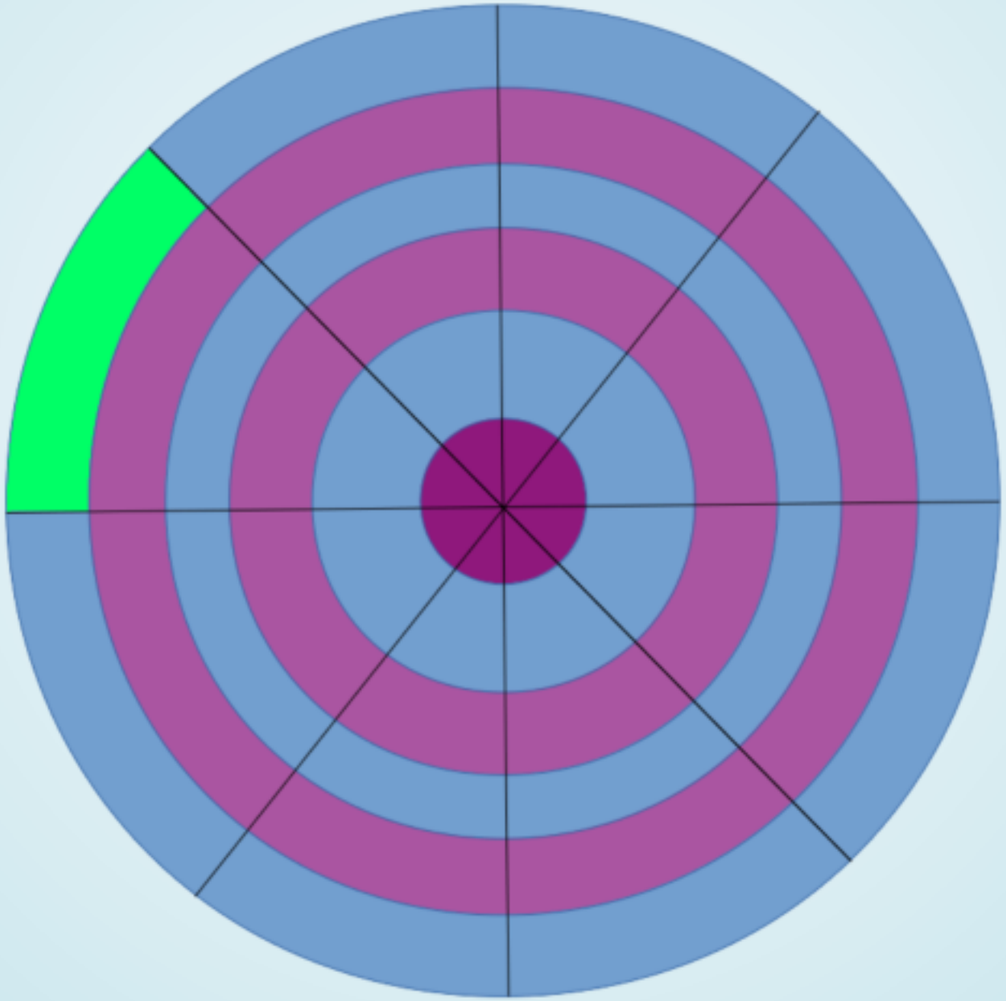
- **2003** Kent Beck: *Green-Bar Patterns: "Fake It", "Preparatory Refactoring"*
- **2009** GOOS's: *Outside-In Design*
- **2013** Emily Bache: *Outside-In development with Double Loop TDD*
- **2013** Ralph Westphal: *Integration Segregation Principle*
- **2014** Justin Searls: *The Failures of "Intro to TDD"*
- **2017** Llewelyn Falco: *Fake it Till you Make It*

BACKWARD CALCULATION

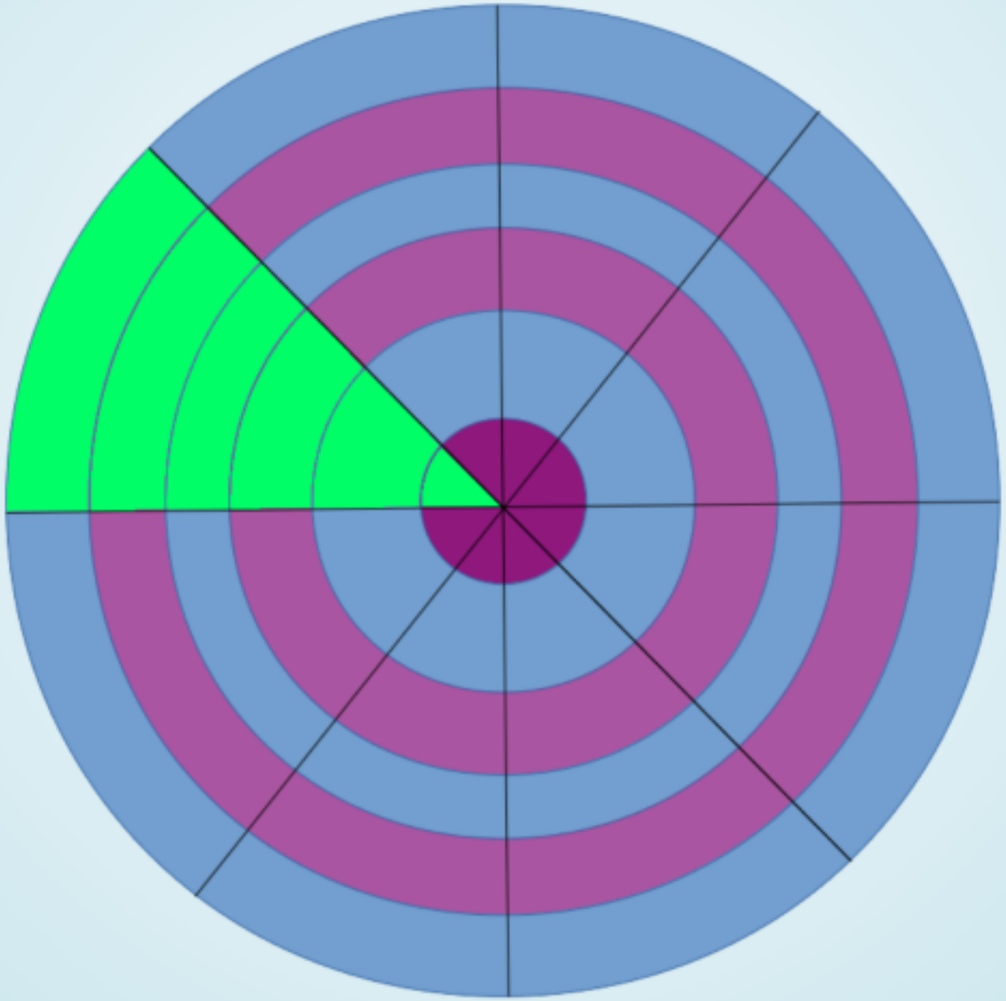
MUNICH



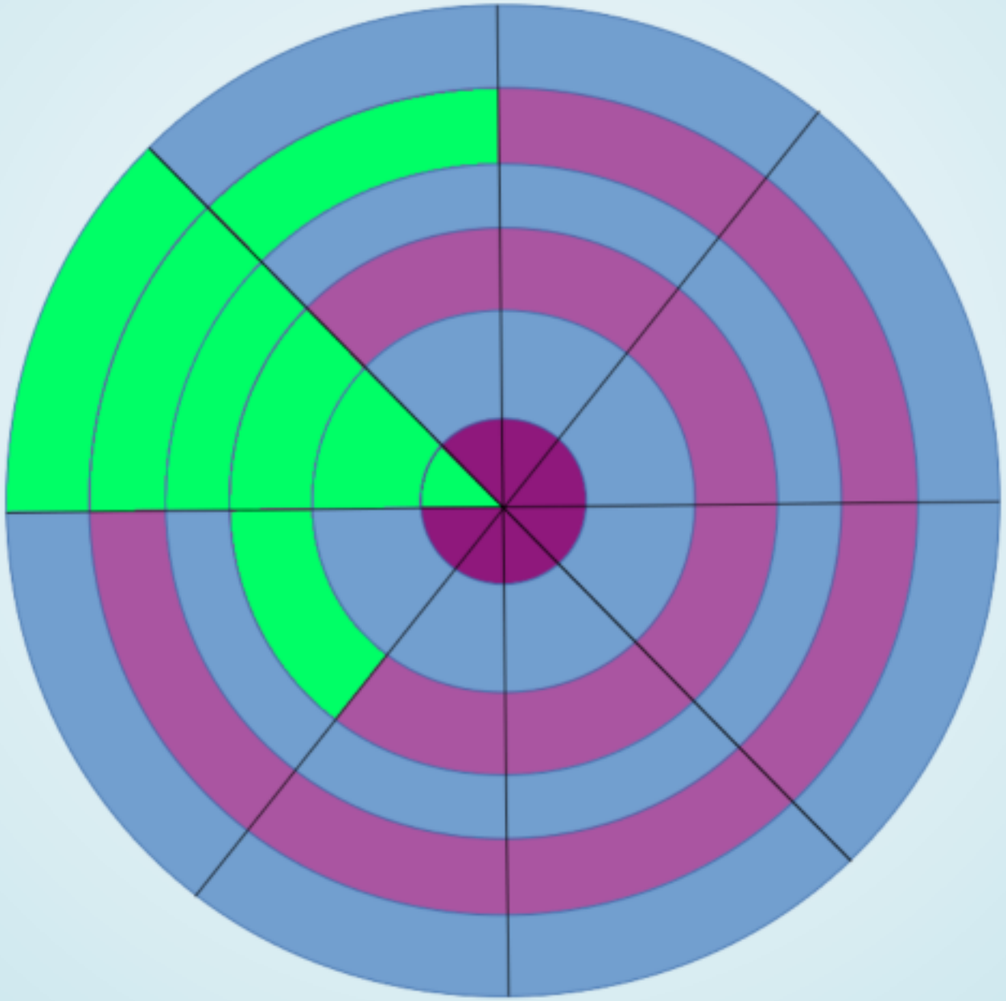
MUNICH



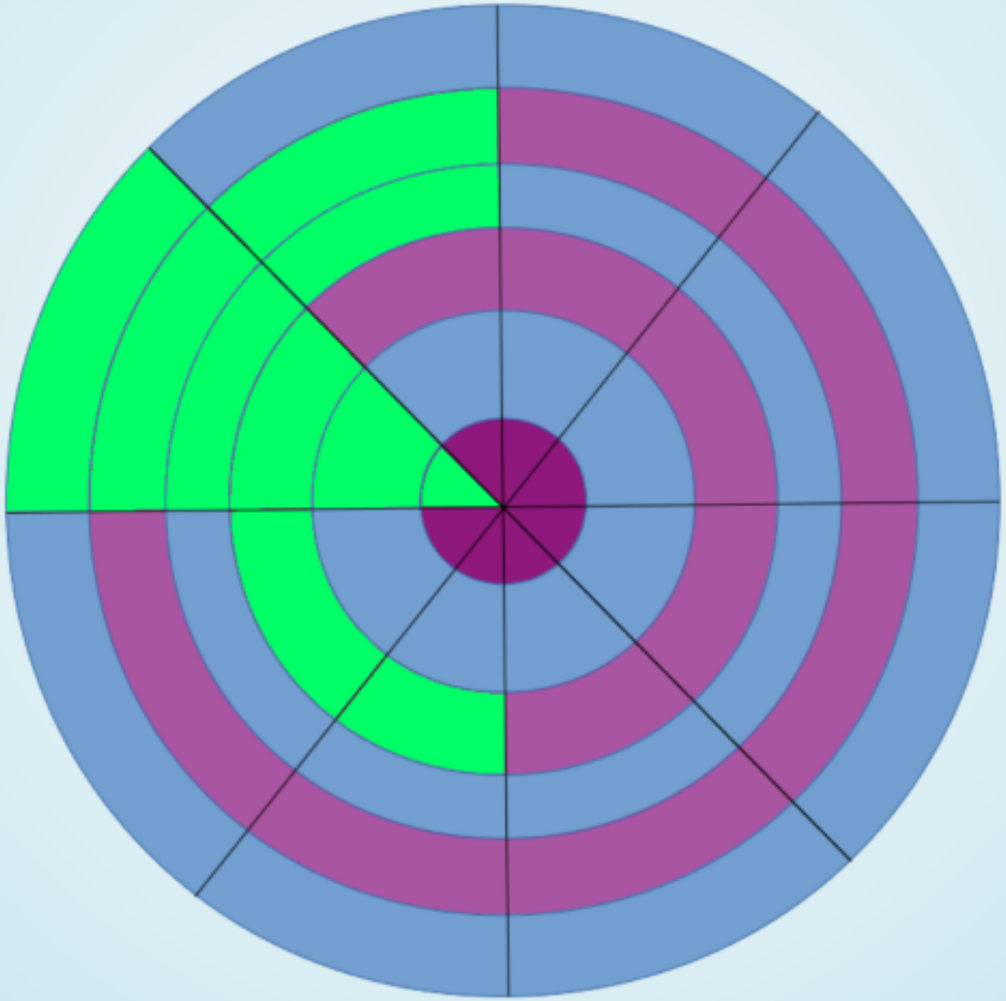
MUNICH



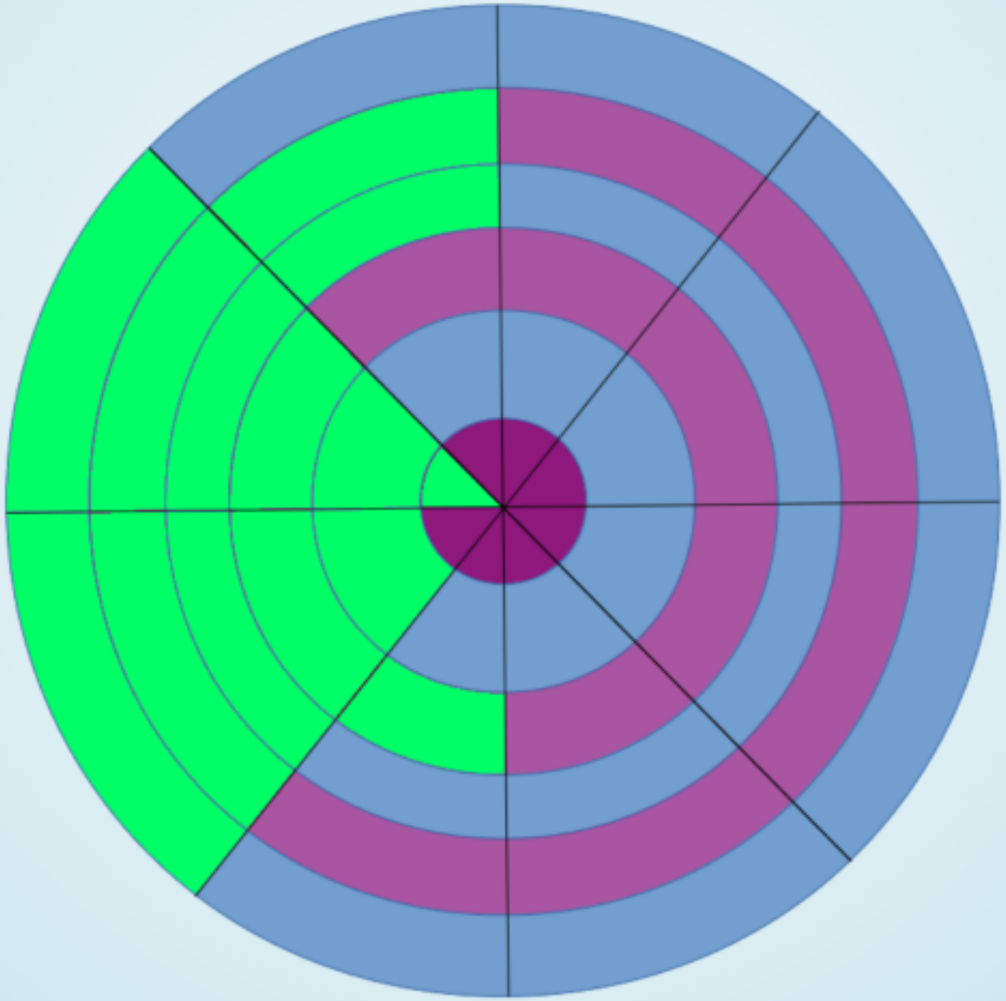
MUNICH



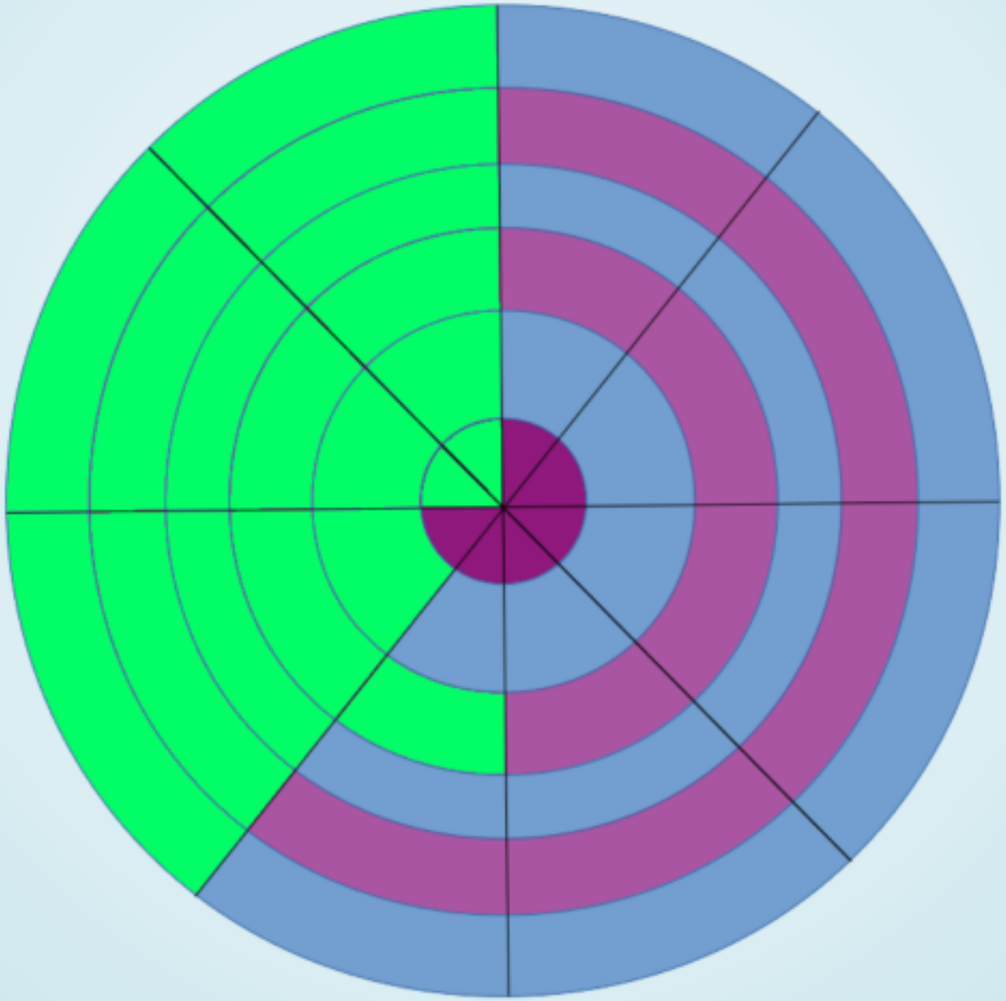
MUNICH



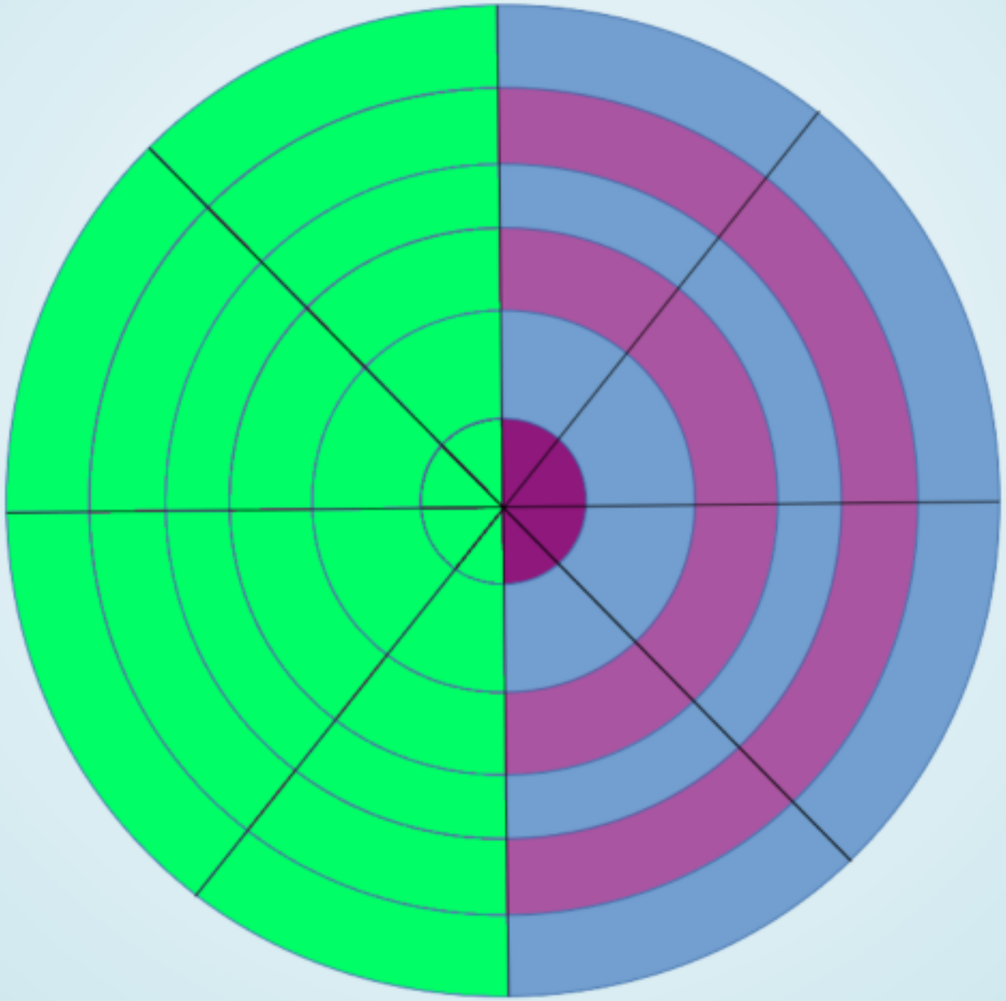
MUNICH



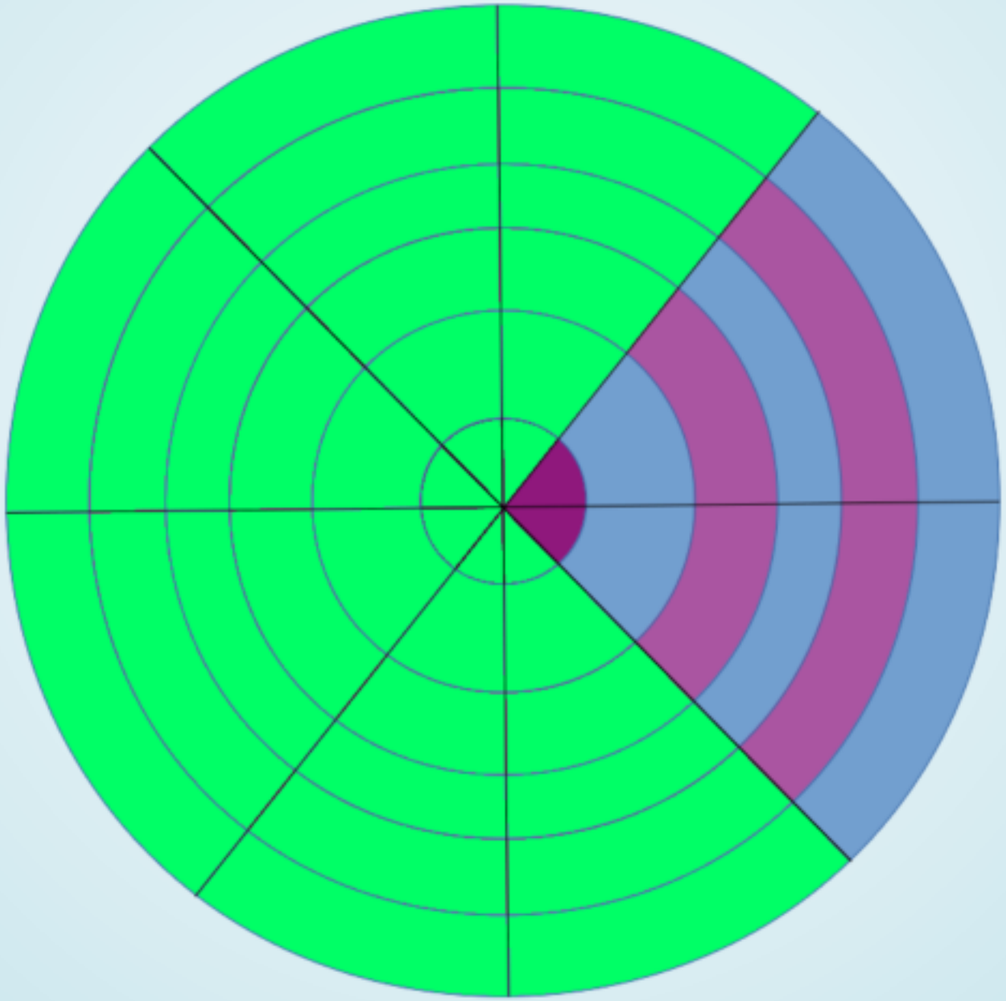
MUNICH



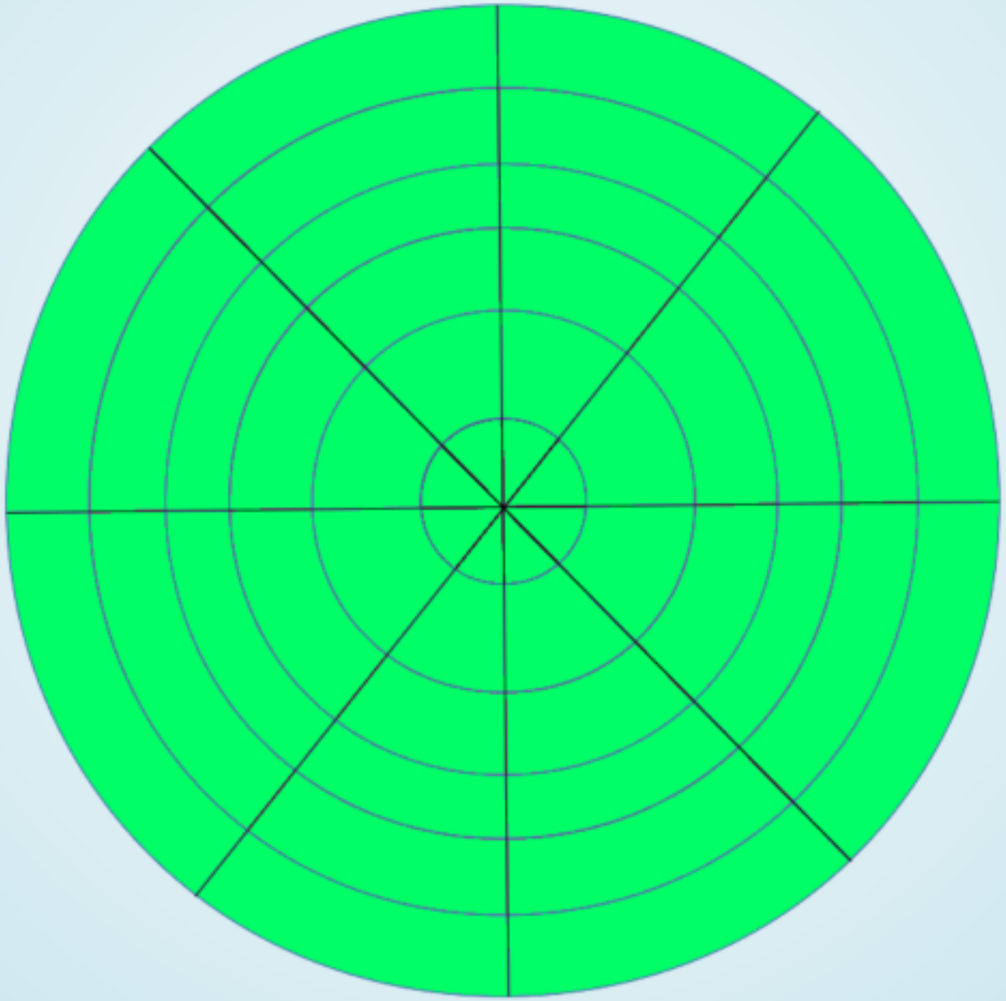
MUNICH



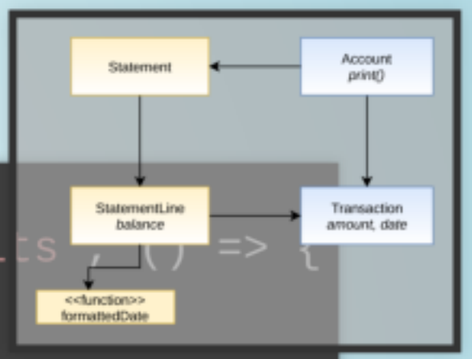
MUNICH



MUNICH



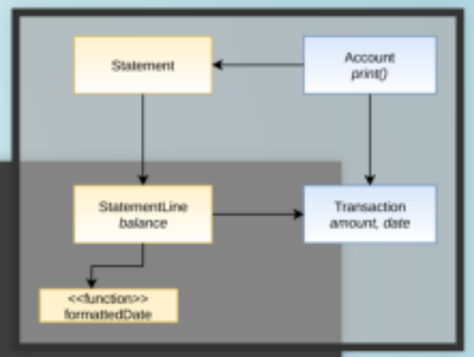
Outer Spec




```
describe('Account', () => {
  it('should print the correct statement after withdraws and deposits', () => {
    const account = new Account();
    account.deposit(new Date(2012, 0, 10), 1000);
    account.deposit(new Date(2012, 0, 13), 2000);

    expect(
      new Statement(account).toString()
    ).toEqual(
      "Date\tAmount\tBalance\n" +
      "10.1.2012\t+1000\t1000\n" +
      "13.1.2012\t+2000\t3000"
    );
  });
});
```

Fake-It

```
class Account {
  deposit(date, amount) {}
}

class Statement {
  constructor(account) {}

  toString() {
    return "Date\tAmount\tBalance\n" +
      "10.1.2012\t+1000\t1000\n" +
      "13.1.2012\t+2000\t3000";
  }
}
```



```
class Statement {
  constructor(account) {}

  toString() {
    return "Date\tAmount\tBalance\n" +
      "10.1.2012\t+1000\t1000\n" +
      "13.1.2012\t+2000\t3000";
  }
}
```

↓ Refactoring: String → Array ↓

```
class Statement {
  constructor(account) {}

  toString() {
    const statementLines = [
      "Date\tAmount\tBalance",
      "10.1.2012\t+1000\t1000",
      "13.1.2012\t+2000\t3000"];
    return statementLines.join('\n')
  }
}
```

```
toString() {
  const statementLines = [
    "Date\tAmount\tBalance",
    "10.1.2012\t+1000\t1000",
    "13.1.2012\t+2000\t3000"];
  return statementLines.join('\n')
}
```

↓ Refactoring: Introduce Constant / Instance Var ↓

```
toString() {
  this.STATEMENT_HEADER = "Date\tAmount\tBalance";
  this.statementLines = [
    "10.1.2012\t+1000\t1000",
    "13.1.2012\t+2000\t3000"
  ];
  const statementLines =
    [this.STATEMENT_HEADER].concat(this.statementLines);
  return statementLines.join('\n')
}
```

```
toString() {
  this.STATEMENT_HEADER = "Date\tAmount\tBalance";
  this.statementLines = [
    "10.1.2012\t+1000\t1000",
    "13.1.2012\t+2000\t3000"
  ];
  const statementLines =
    [this.STATEMENT_HEADER].concat(this.statementLines);
  return statementLines.join('\n')
}
```

↓ Refactoring: Move Assignment to Constructor ↓

```
constructor(account) {
  this.STATEMENT_HEADER = "Date\tAmount\tBalance";
  this.statementLines = [
    "10.1.2012\t+1000\t1000",
    "13.1.2012\t+2000\t3000"
  ]
}
toString() {
  const statementLines =
    [this.STATEMENT_HEADER].concat(this.statementLines);
  return statementLines.join('\n')
}
```

```
  toString() {
    const statementLines =
      [this.STATEMENT_HEADER].concat(this.statementLines);
    return statementLines.join('\n')
  }
```

↓ Refactoring: Inline Variable ↓

```
class Statement {
  ...
  toString() {
    return [this.STATEMENT_HEADER]
      .concat(this.statementLines)
      .join('\n');
  }
}
```

```
  constructor(account) {
    this.STATEMENT_HEADER = "Date\tAmount\tBalance";
    this.statementLines = [
      "10.1.2012\t+1000\t1000",
      "13.1.2012\t+2000\t3000"
    ]
  }
```

↓ Refactoring: Introduce Class ↓

```
  constructor(account) {
    this.STATEMENT_HEADER = "Date\tAmount\tBalance";
    this.statementLines = [
      new StatementLine("10.1.2012\t+1000\t1000"),
      new StatementLine("13.1.2012\t+2000\t3000")
    ]
  }
...

class StatementLine {
  constructor(line)  {this.line = line;}
  toString() {return this.line;}
}
```

INTERMEDIATE TESTS

```
describe('StatementLine#toString', () => {
  it('should return a tab-separated String', () => {
    const transaction = new Transaction("21.2.2012", "+1000");
    const statementLine = new StatementLine(transaction, 4000);
    expect( statementLine.toString() ).toEqual(
      "21.2.2012\t+1000\t4000"
    );
  });
});
```

```
class StatementLine {
  constructor(transaction, balance)  {
    this.transaction = transaction;
    this.balance = balance;
  }
  toString() {
    return [
      this.transaction.date,
      this.transaction.amount,
      this.balance
    ].join('\t');
  }
}
```

```
class Transaction {
  constructor(date, amount) {
    this.date = date;
    this.amount = amount;
  }
}
```

```
class Statement {
  constructor(account) {
    this.STATEMENT_HEADER = "Date\tAmount\tBalance";
    this.statementLines = [
      new StatementLine("10.1.2012\t+1000\t1000"),
      new StatementLine("13.1.2012\t+2000\t3000")
    ]
  }
  ...
```

↓ Refactoring: Split Arguments ↓

```
class Statement {
  constructor(account) {
    this.STATEMENT_HEADER = "Date\tAmount\tBalance";
    this.statementLines = [
      new StatementLine(new Transaction("10.1.2012", "+1000"), 1000),
      new StatementLine(new Transaction("13.1.2012", "+2000"), 3000)
    ]
  }
  ...
```

```
class Statement {
  constructor(account) {
    this.STATEMENT_HEADER = "Date\tAmount\tBalance";
    this.statementLines = [
      new StatementLine(new Transaction("10.1.2012", "+1000"), 1000),
      new StatementLine(new Transaction("13.1.2012", "+2000"), 3000)
  ...
```

↓ Refactoring: Move Transactions to Account ↓

```
class Account {
  constructor()  {
    this.transactions = [
      new Transaction("10.1.2012", "+1000"),
      new Transaction("13.1.2012", "+2000")
...
class Statement {
  constructor(account) {
    this.STATEMENT_HEADER = "Date\tAmount\tBalance";
    this.statementLines = [
      new StatementLine(account.transactions[0], 1000),
      new StatementLine(account.transactions[1], 3000)
    ]
  }
```

```
class Account {
  constructor()  {
    this.transactions = [
      new Transaction("10.1.2012", "+1000"),
      new Transaction("13.1.2012", "+2000")
...
```

↓ Refactoring: Replace Fake Value ↓

```
class Account {
  constructor()  {
    this.transactions = []
  }
  deposit(date, amount) {
    this.transactions.push(new Transaction(formattedDate(date), "+" + amount))
  }
}
```

```
class Statement {
  constructor(account) {
    this.STATEMENT_HEADER = "Date\tAmount\tBalance";
    this.statementLines = [
      new StatementLine(account.transactions[0], 1000),
      new StatementLine(account.transactions[1], 3000)
    ]
  }
  ...
```

↓ Refactoring: Replace Fixed Index with map ↓

```
class Statement {
  constructor(account) {
    this.STATEMENT_HEADER = "Date\tAmount\tBalance";
    this.statementLines = account.transactions.map(
      (t, i, ts) => new StatementLine(t, balance(ts, i))
    );
  }
```

MUNICH CONSEQUENCES

- + No YAGNI
- + Obvious next step
- + ...from many choices
- - ...from many choices
- + No Mocks needed!!!
- + Higher Amount of Time in Green
- -> Lower Risk

BEST SCHOOL?

> *It depends*™
>
> — any Consultant, ever


CRAFTSMAN
NICKEL PLATED
CHOOSE
BEST TOOL
FOR THE JOB


WILL IT BLEND?


EXERCISE
WWW.CYBER-DOJO.ORG

REFERENCES: GENERAL

- Interview with Kent Beck
- Martin Fowler: Mocks aren't Stubs
- Martin Fowler: Solitary or Sociable
- David Völkel: Mockist vs Classicists - Slides
- Book ATDD by Markus Gärtner

REFERENCES: CHICAGO

- J.B. Rainsberger: Integrated Tests are a Scam
- J.B. Rainsberger: Universal Architecture
- Justin Seals: Detroit school TDD
- Book: Test-Driven Development by Kent Beck

REFERENCES: LONDON

- Emily Bache: Double-Loop TDD
- Book: Growing Object Oriented Software Guided by Tests by Steve Freeman & Nat Price

REFERENCES: MUNICH

- David Völkel: Fake-it Outside-In
- David Völkel: Screencasts (Youtube)

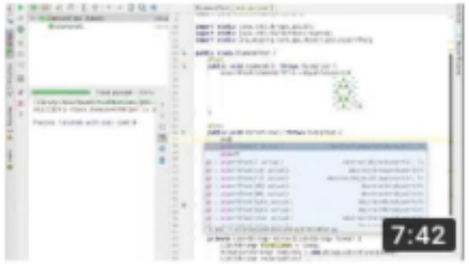
David Völkel

15 subscribers

HOME

Uploads

PLAY ALL

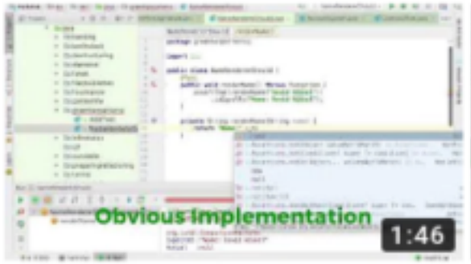


Fake it Outside-In TDD

171 views • 1 year ago

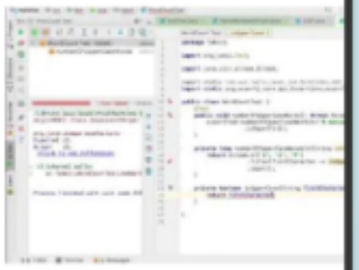
Preparatory Refactorings with TDD

106 views • 1 year ago



Kent Beck's Greenbar Patterns

229 views • 1 year ago



Fake it TDD: Working backwards from an...

103 views • 1 year ago

IMAGE CREDITS

- Kent Beck by Improve It on Flickr, CC BY-SA 2.0
- Making of Harry Potter - Karen Roe on Flickr, CC BY 2.0
- Children practicing Taekwondo in Kathmandu, Nepal by Adli Wahid on Unsplash, CC0
- Watch Where You're Putting That Thing: Chris Gilmore Follow, CC BY-SA 2.0
- Map images from Googlemaps
- Drunken Kermit by Alexas Fotos on Pixabay, CC0
- Skyline Skyscraper by PIRO4D on Pixabay, CC0
- Detroit Photo by Sawyer Bengtson on Unsplash, CC0
- London Photo by Luca Micheli on Unsplash, CC0
- Munich Photo by Christoph Keil on Unsplash, CC0
- Lego Photo by Rick Mason on Unsplash, CC0
- Aerial view Zhangjiajie glass bridge by www.highestbridges.com
- Fire Motorcycle Stunt by digihanger on Pixabay, CC0
- Trophy by Fauzan Saari on Unsplash, CC0
- iPhone in Blender from Blendtec's Will It Blend? https://www.youtube.com/watch?v=lBUJcD6Ws6s

THX

@thbrunzendorf
@MarcoEmrich