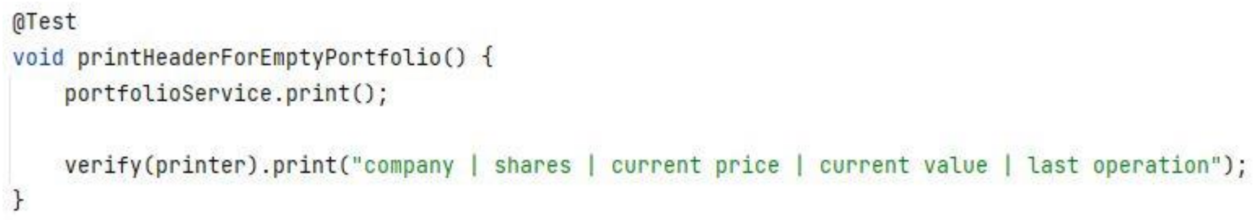
REVIEW

A FLY THROUGH THE COURSE

10.11.2021

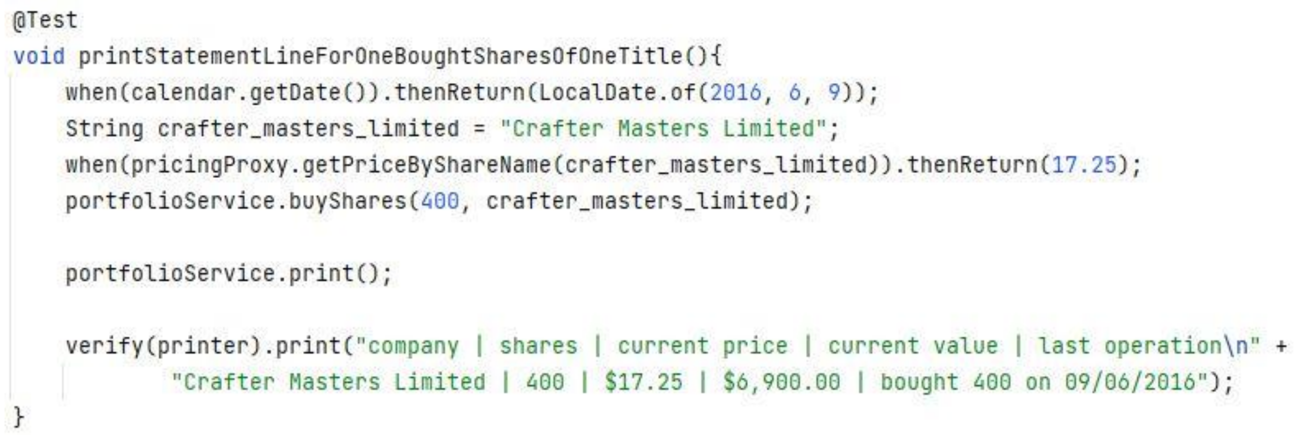
WALKING

Baby Steps



```
@Test
void printHeaderForEmptyPortfolio() {
    portfolioService.print();

    verify(printer).print("company | shares | current price | current value | last operation");
}
```

Testing behaviour



```
@Test
void printStatementLineForOneBoughtSharesOfOneTitle(){
    when(calendar.getDate()).thenReturn(LocalDate.of(2016, 6, 9));
    String crafter_masters_limited = "Crafter Masters Limited";
    when(pricingProxy.getPriceByShareName(crafter_masters_limited)).thenReturn(17.25);
    portfolioService.buyShares(400, crafter_masters_limited);

    portfolioService.print();

    verify(printer).print("company | shares | current price | current value | last operation\n" +
            "Crafter Masters Limited | 400 | $17.25 | $6,900.00 | bought 400 on 09/06/2016");
}
```

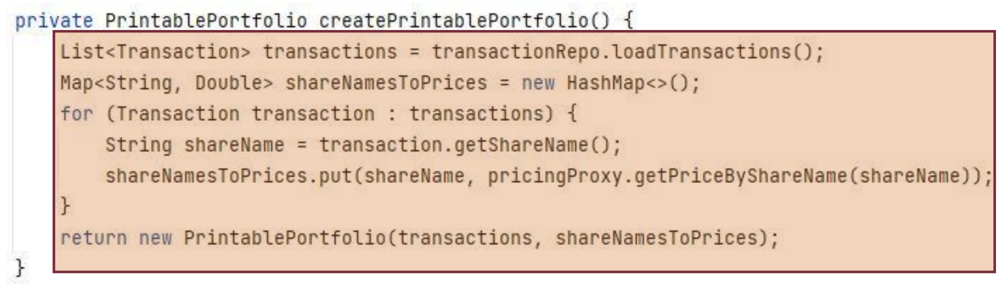
Triangulation without changing **dimensions of freedom** too early

- first only proceeded in buy Shares (first just one line, then 2)
- then only in sell Shares
- Combined input at last

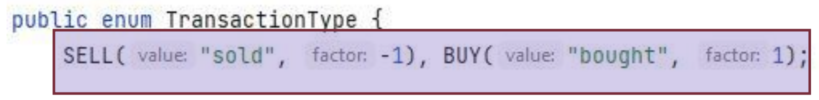
Red – Green – Refactor

- With baby steps
- Test behaviour and not implementation details
- Use triangulation
- One dimension of freedom after the other

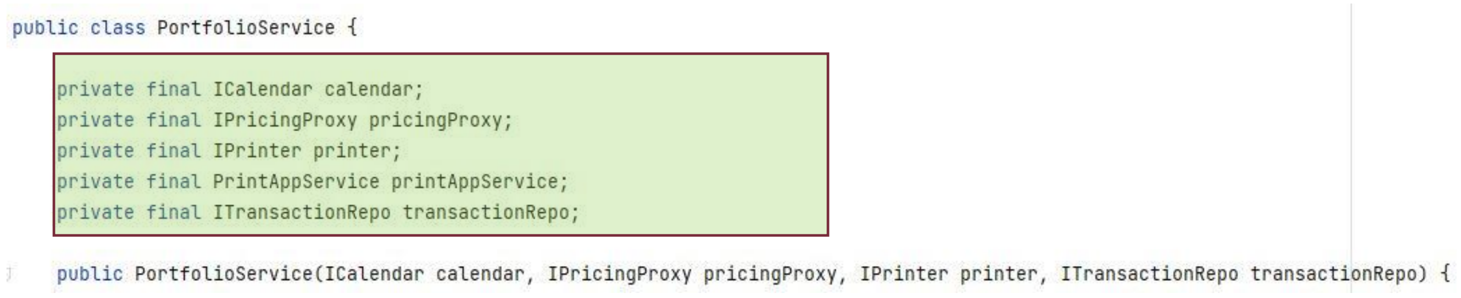
WALKING – OBJECT CALISTHENICS



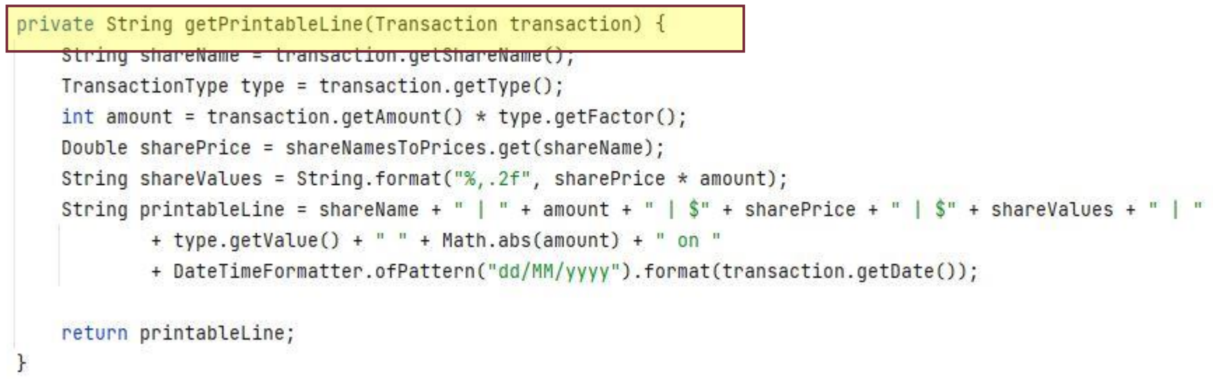
```
private PrintablePortfolio createPrintablePortfolio() {
    List<Transaction> transactions = transactionRepo.loadTransactions();
    Map<String, Double> shareNamesToPrices = new HashMap<>();
    for (Transaction transaction : transactions) {
        String shareName = transaction.getShareName();
        shareNamesToPrices.put(shareName, pricingProxy.getPriceByShareName(shareName));
    }
    return new PrintablePortfolio(transactions, shareNamesToPrices);
}
```



```
public enum TransactionType {
    SELL( value: "sold",  factor: -1), BUY( value: "bought",  factor: 1);
```



```
public class PortfolioService {

    private final ICalendar calendar;
    private final IPricingProxy pricingProxy;
    private final IPrinter printer;
    private final PrintAppService printAppService;
    private final ITransactionRepo transactionRepo;

    public PortfolioService(ICalendar calendar, IPricingProxy pricingProxy, IPrinter printer, ITransactionRepo transactionRepo) {
```



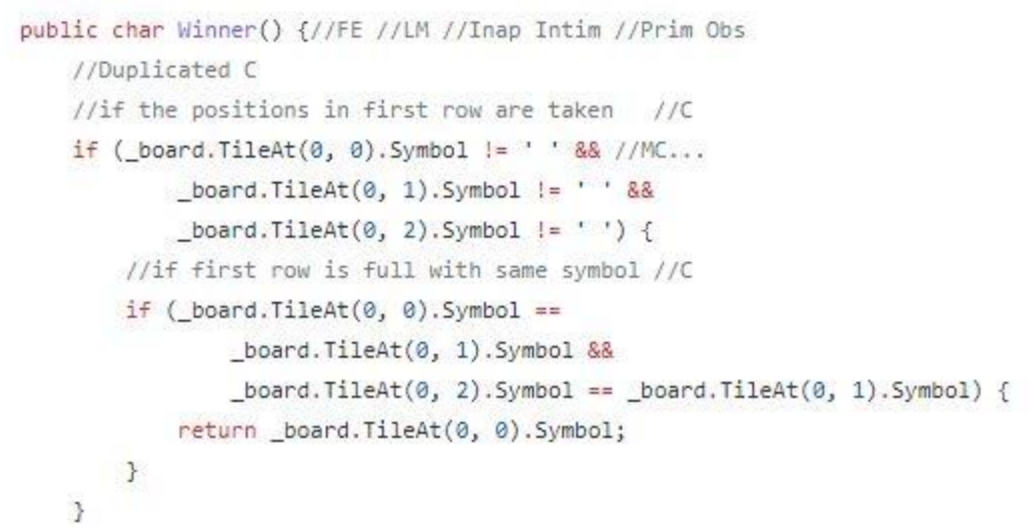
```
private String getPrintableLine(Transaction transaction) {
    String shareName = transaction.getShareName();
    TransactionType type = transaction.getType();
    int amount = transaction.getAmount() * type.getFactor();
    Double sharePrice = shareNamesToPrices.get(shareName);
    String shareValues = String.format("%,.2f", sharePrice * amount);
    String printableLine = shareName + " | " + amount + " | $" + sharePrice + " | $" + shareValues + " | "
            + type.getValue() + " " + Math.abs(amount) + " on "
            + DateTimeFormatter.ofPattern("dd/MM/yyyy").format(transaction.getDate());

    return printableLine;
}
```

Object Calisthenics

- Only one level of indentation
- Wrap primitives and Strings
- Only One dot per line

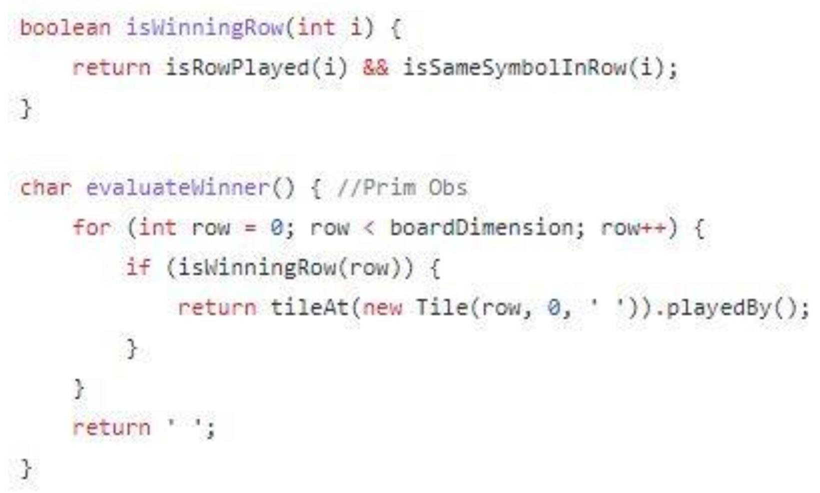
- No more than 2 instance variables per class
- No public getters/setters/properties

RUNNING – CODE SMELLS



```
public char Winner() {//FE //LM //Inap Intim //Prim Obs
    //Duplicated C
    //if the positions in first row are taken   //C
    if (_board.TileAt(0, 0).Symbol != ' ' && //MC...
            _board.TileAt(0, 1).Symbol != ' ' &&
            _board.TileAt(0, 2).Symbol != ' ') {
        //if first row is full with same symbol //C
        if (_board.TileAt(0, 0).Symbol ==
                _board.TileAt(0, 1).Symbol &&
                _board.TileAt(0, 2).Symbol == _board.TileAt(0, 1).Symbol) {
            return _board.TileAt(0, 0).Symbol;
        }
    }
```

…



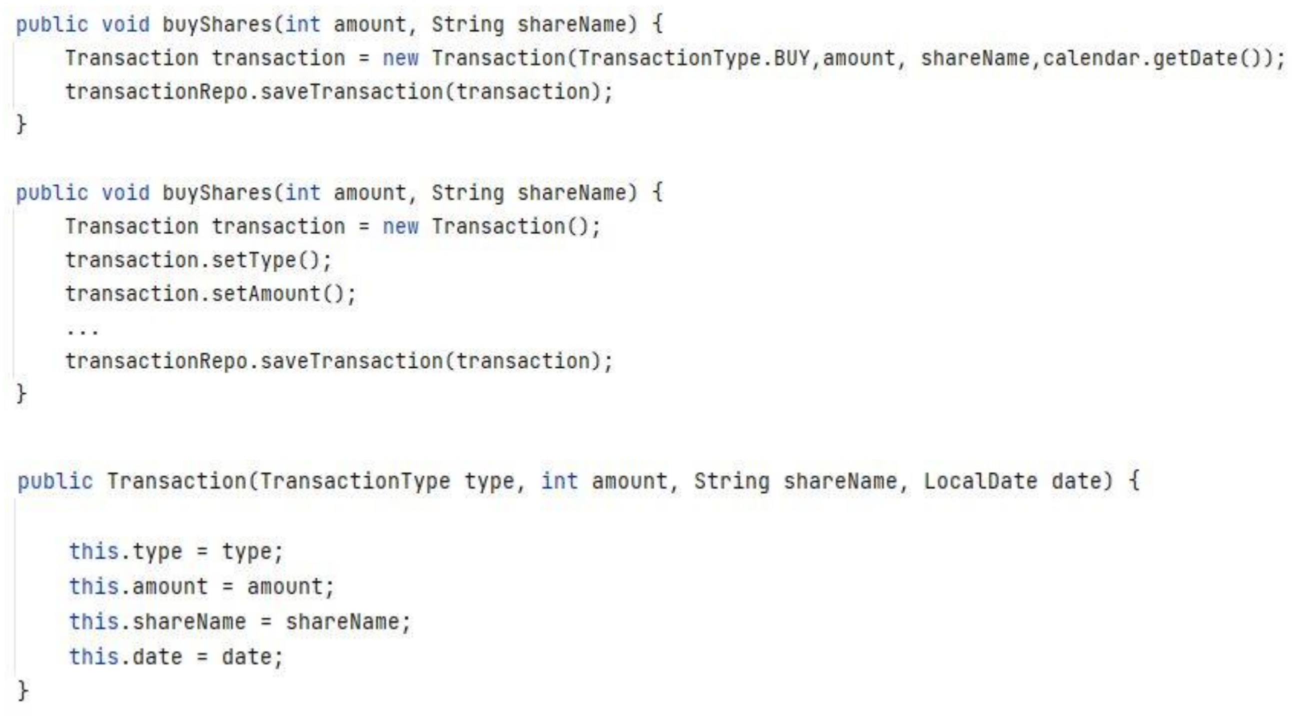
```
boolean isWinningRow(int i) {
    return isRowPlayed(i) && isSameSymbolInRow(i);
}

char evaluateWinner() { //Prim Obs
    for (int row = 0; row < boardDimension; row++) {
        if (isWinningRow(row)) {
            return tileAt(new Tile(row, 0, ' ')).playedBy();
        }
    }
    return ' ';
}
```

We made it much more beautiful!

- Feature Envy: method in class Game uses methods of class Board exessively
- Long method
- Inappropriate Intimicy: method uses internal Field « Symbol » of class Board
- Duplicated Code
- Comments (even wrong ones!)
- Primitive Obsession

Start refactoring based on the 80-20 rule!

FLYING - CONNASCENCE



```java
public void buyShares(int amount, String shareName) {
    Transaction transaction = new Transaction(TransactionType.BUY,amount, shareName,calendar.getDate());
    transactionRepo.saveTransaction(transaction);
}

public void buyShares(int amount, String shareName) {
    Transaction transaction = new Transaction();
    transaction.setType();
    transaction.setAmount();
    ...
    transactionRepo.saveTransaction(transaction);
}

public Transaction(TransactionType type, int amount, String shareName, LocalDate date) {

    this.type = type;
    this.amount = amount;
    this.shareName = shareName;
    this.date = date;
}
```

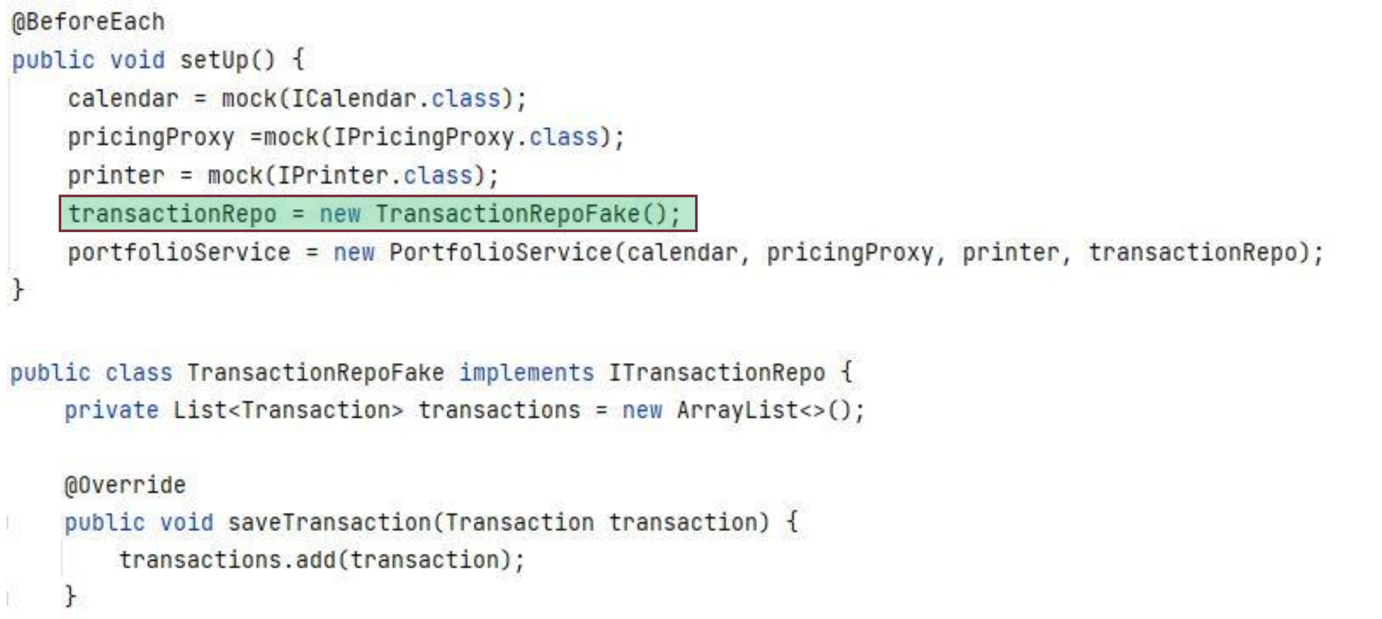
Dynamic

- Execution order

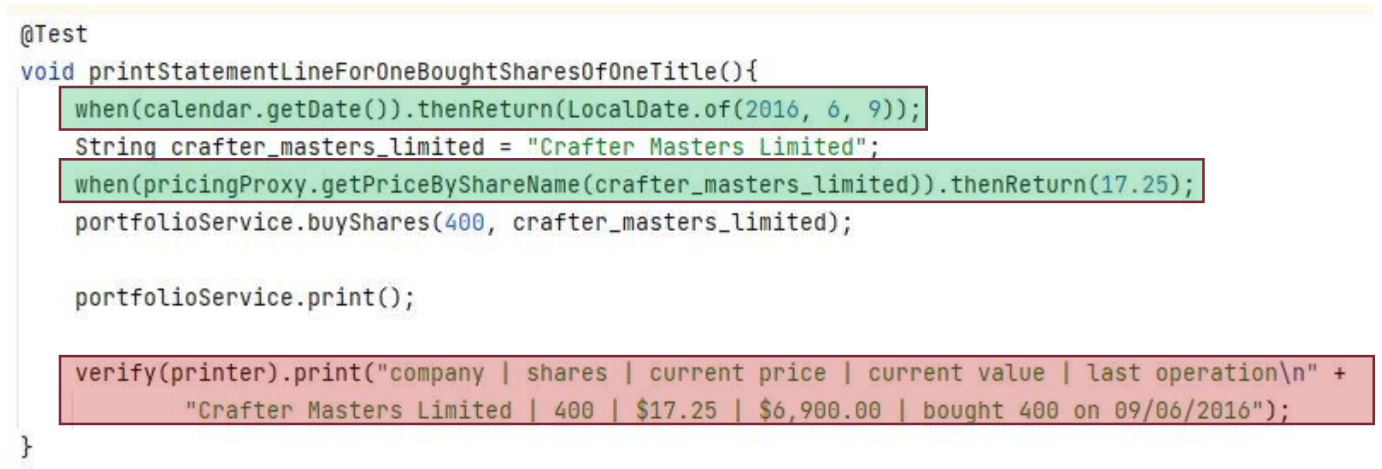
Static

- Position
- Name
- Type
- Would accur when we would not wrap the transaction type in an enum

FLYING – TEST DOUBLES



```
@BeforeEach
public void setUp() {
    calendar = mock(ICalendar.class);
    pricingProxy =mock(IPricingProxy.class);
    printer = mock(IPrinter.class);
    transactionRepo = new TransactionRepoFake();
    portfolioService = new PortfolioService(calendar, pricingProxy, printer, transactionRepo);
}

public class TransactionRepoFake implements ITransactionRepo {
    private List<Transaction> transactions = new ArrayList<>();

    @Override
    public void saveTransaction(Transaction transaction) {
        transactions.add(transaction);
    }
```



```
@Test
void printStatementLineForOneBoughtSharesOfOneTitle(){
    when(calendar.getDate()).thenReturn(LocalDate.of(2016, 6, 9));
    String crafter_masters_limited = "Crafter Masters Limited";
    when(pricingProxy.getPriceByShareName(crafter_masters_limited)).thenReturn(17.25);
    portfolioService.buyShares(400, crafter_masters_limited);

    portfolioService.print();

    verify(printer).print("company | shares | current price | current value | last operation\n" +
            "Crafter Masters Limited | 400 | $17.25 | $6,900.00 | bought 400 on 09/06/2016");
}
```

Commands

- Mocks or Spies

Queries

- Stubs or Fakes

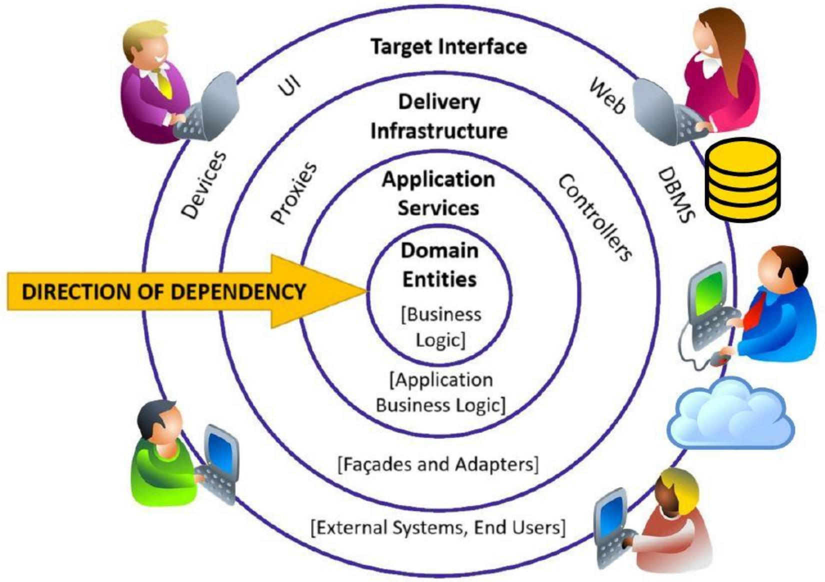
Framework or handmade?

DIRECTION OF DEPENDENCY

Begin from external Point of View

- Start from Acceptance Test

- Acceptance Test helps for triangulation and finding the next “baby step”
  - When we finished 1 degree of freedom the acceptance test pointed us to the next
- No Dependencies from the inside to the outer world:
  - Use Interfaces -> Implementation of outer Systems can easily be changed
  - Test Doubles help testing and implementing the domain and application layer services



[Agile Technical Practices Distilled. Pedro M. Santos, Marco Consolaro, Alessandro Di Gioia]

LEARNINGS AND QUESTIONS

Design upfront helps, but can also block when not correct (like I was implementing the Bowling Kata)

- Mob helps when blocked

Refactor constantly and aggressively

- There will always be improvements
- But don’t change things that don’t need to be changed
  - Refactor when you have to add new features
  - Create technical dept tasks for things you find, but are not urgent?

What did we learn from Mob Programming?

- Helps to learn from each other
- Helps to get to know each other better
- Team-building
- Builds up trust in each other
  - Better communication for constructive criticism and improvement ideas
- What do you think?

Any other questions or additions?

MERCI

Looking forward to next course!
And
Have a great weekend!

LinkedIn: Dominique Latza

dominique_latza@live.de