AKADEMIA FINANSÓW I BIZNESU VISTULA

WSTĘP DO PROGRAMOWANIA DLA TESTERÓW

„Everything I was I carry with me, everything I will be, lies waiting on the road ahead”

by Ma Jian

Wstęp do programowania Java dla testerów

Remigiusz Dudek

AGENDA

„THE FIRST STEP TOWARDS GETTING SOMEWHERE IS TO DECIDE THAT YOU ARE NOT GOING TO STAY WHERE YOU ARE" BY ANNONYMOUS

- Day 1 (Basics + Java basics)
  - IDE – project structure
  - Class/Object/Package
  - First @Test
  - Basics (variables / methods)
  - Primitive types
  - Assetions
  - Basic Classes
  - Basic inheritance / Object creation
  - Equality
  - Strings
  - Arrays/Collection
  - Steer the flow (conditions/loops)
- Day 2 (OO Design)
  - Data driven testing (Parameters & File IO)
  - Inheritance
  - Polymorphism
  - Page Object Pattern
  - Data driven testing
  - Exceptions
- Day 3 (Advanced concepts)
  - F.I.R.S.T.
  - Clean code
  - S.O.L.I.D.
  - Test frameworks

AKADEMIA FINANSÓW I BIZNESU VISTULA

F.I.R.S.T.

„We all need feedback, that's how we improve"

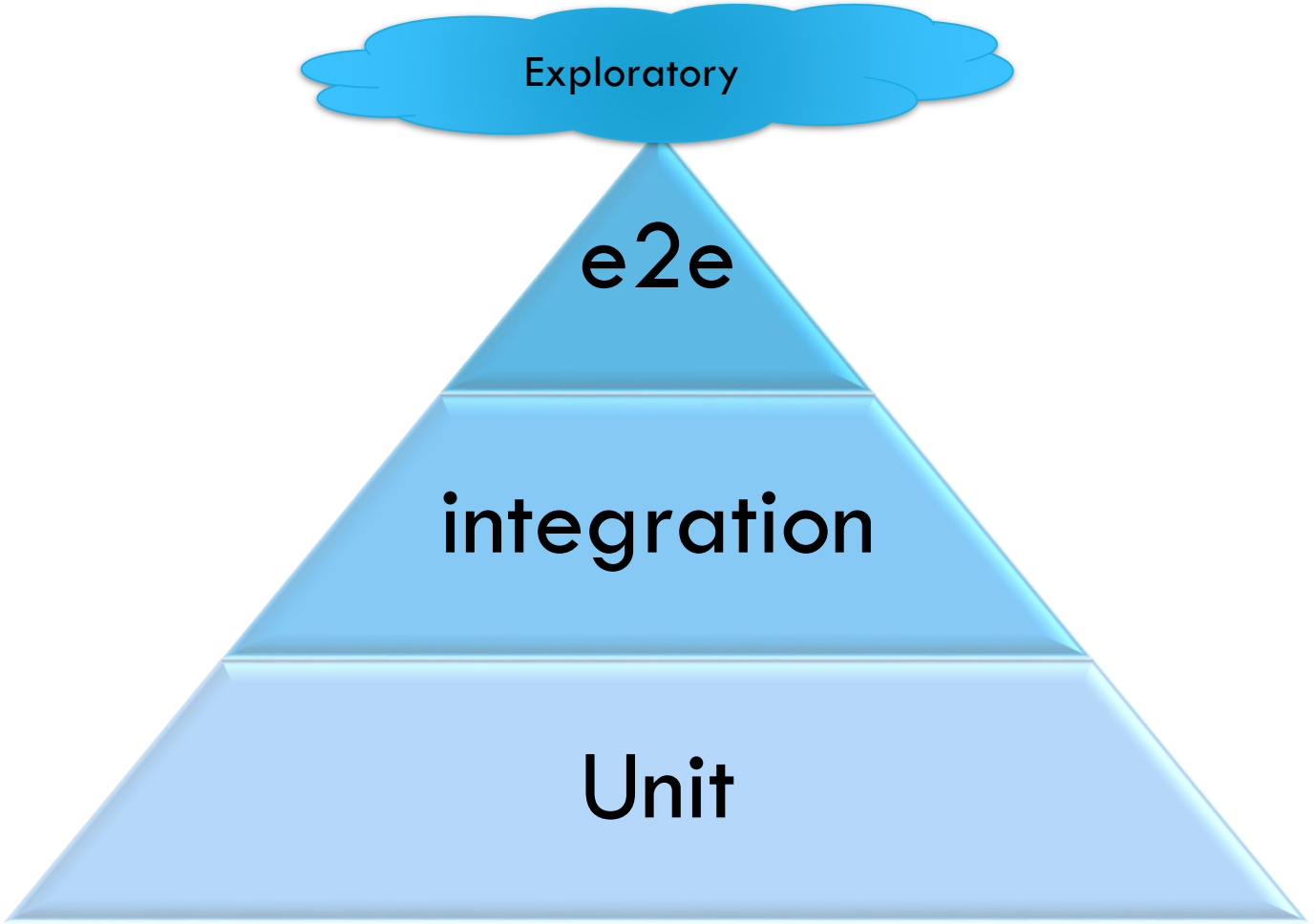
by Bill Gates

Wstęp do programowania Java dla testerów

Remigiusz Dudek

TEST AUTOMATION PYRAMID

„FOR EVERY MINUTE SPENT ORGANIZING, AN HOUR IS EARNED" BY BENJAMIN FRANKLIN



FAST

„*TEST FAST, FAIL FAST, ADJUST FAST*" BY TOM PETERS

- Duration
  - milliseconds
  - Second long test is unacceptable long test
  - Quarter-second long test is a painfully long test
  - Setup and tear down counts as well
- if you hesitate to run your tests after each change they are too slow
- how to test that something is not happening?
- how about end to end tests?

ISOLATED/INDEPENDENT

„HAPPINESS OR SORROW, WHATEVER BEFALLS YOU, WALK ON UNTOUCHED, UNATACHED" BY BUDDHA

- Idempotent and safe
- Clean before not after
- No specific order of tests
- No basing on output of other test

REPEATABLE

„FAILING TEST IS NOT OCCASIONAL ACT, IT'S A CONSTANT ATTITUDE" BY M.L.KING (ALMOST)

- credible
- do not depend on external services

SELF-VALIDATING

„*ONE CANNOT DENY, WHAT'S OBVIOUS TO SEE*" BY ANNONYMOUS

- clear pass or fail
- no analysis

TIMELY

„*THE TIME IS ALWAYS RIGHT TO DO WHAT IS RIGHT*" BY M. L. KING

- just before code is written

AKADEMIA FINANSÓW I BIZNESU VISTULA

CLEAN CODE

„A clean place is a safe place"

Wstęp do programowania Java dla testerów

Remigiusz Dudek

NAMES

„*WORDS ARE POWERFULL, THEY CAN CREATE OR THEY CAN DESTROY*" BY ANNONYMOUS

- intention revealing
- have searching in mind
- ubiquities language
- add context if needed (additional private method)

FUNCTIONS

„*DO ONE THING, BUT DO IT RIGHT*" BY ANNONYMOUS

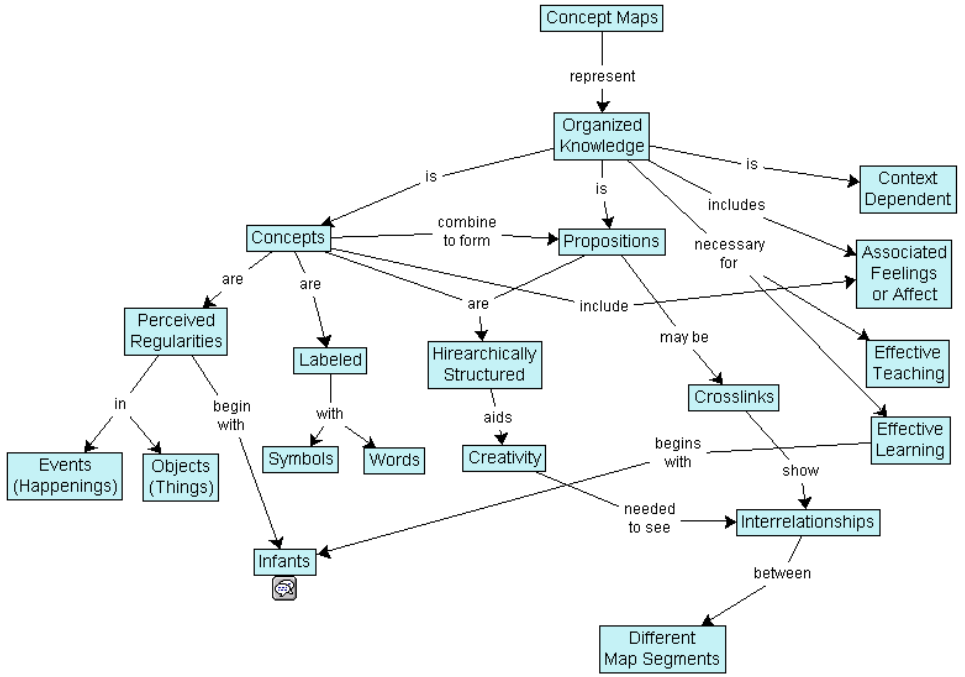
- do one thing
- one level of abstraction
- read code like book (top-down)
- beware of boolean arguments
- as few parameters as possible
- returned object should be obvious
- no side effect
- extract try/catch
- DRY

$$A = \pi r^2$$

DATA STRUCTURES AND CLASSES

„*DO ONE THING, BUT DO IT RIGHT*" BY ANNONYMOUS

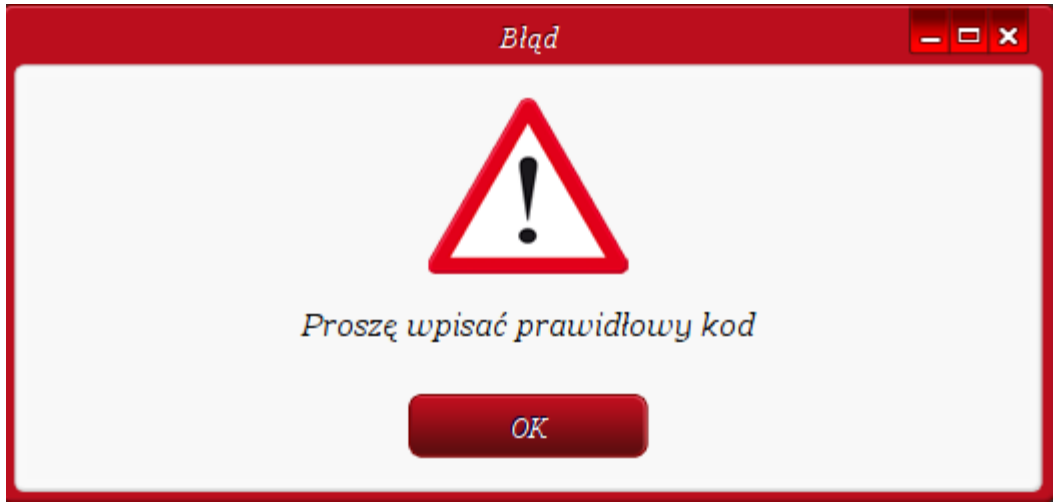
- Law of Demeter (aka train wrecks)
- Encapsulation



ERROR HANDLING

„AN ERROR DOES NOT BECOME A MISTAKE, UNTIL YOU REFUSE TO CORRECT IT" BY JOHN KENNEDY

- use exceptions for handling errorous situations
- use unchecked exceptions
- name exceptions intuitively
- define exception in terms of business problems (not technical problems)
- dont return null, don't pass null – it's always asking for trouble



CLASSES

„*I COULD EITHER WATCH IT HAPPEN OR BE PART OF IT*" BY ELON MUSK

- encapsulation
- small
- SRP
- cohesion
- depend upon an abstraction

AKADEMIA FINANSÓW I BIZNESU VISTULA

S.O.L.I.D.

„In matters of style swim with the current,

In matters of principles stand like a rock”

By Thomas Jefferson

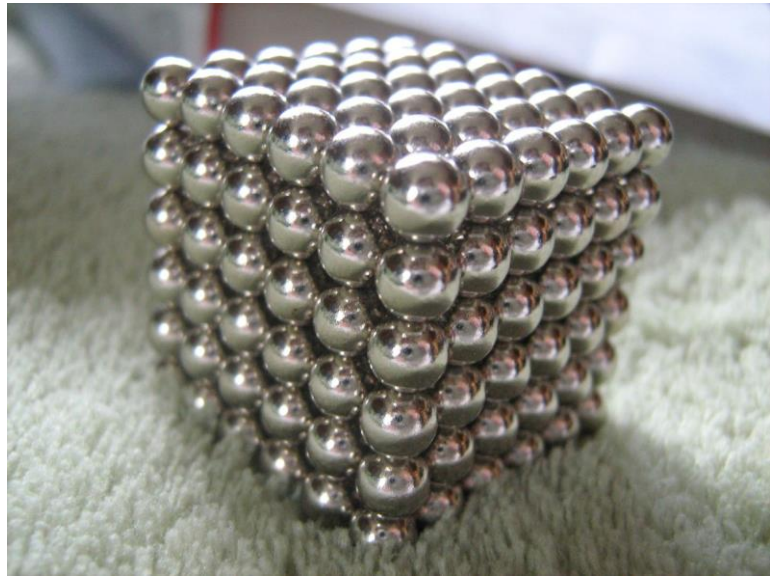
Wstęp do programowania Java dla testerów

Remigiusz Dudek

SINGLE RESPONSIBILITY PRINCIPLE

„*DO ONE THING AND DO IT RIGHT*" BY ANNOUNYMOUS

- you should be changing a class for one reason only
- cohesion may be a hint



OPEN/CLOSE PRINCIPLE

„CLASS SHOULD BE OPEN FOR EXTENSION , BUT CLOSED FOR MODIFICATION"

- software design should support adding new features with minimal changes to the existing code
- consequence of applying the principle where it should not be applied is unnecessary abstraction and increased complexity
- consequence of applying the principle where it should be applied is a clear picture of high level proces and additionally it is possible to add new features without modifying already tested code

LISKOV SUBSTITUTION PRINCIPLE

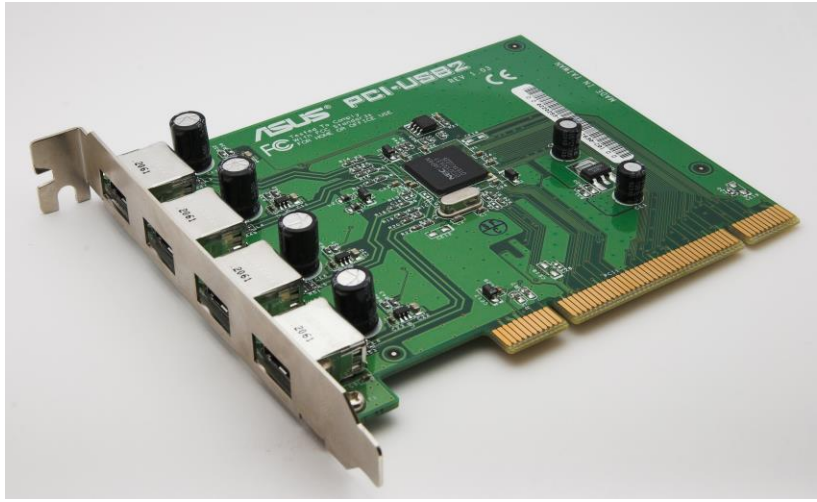
„IT IS SQUARE THAT IS SPECIAL RECTANGLE NOT THE OTHER WAY ROUND"

- Derived classes should not change the behaviour of a base class (they should extend it)
- Whenever we use a given base type in a program it should be safe to replace it with any of it's subtypes

INTERFACE SEGREGATION

„AN INTERFACE IS LIKE A JOKE, IF YOU NEED TO EXPLAIN IT, ITS NOT GOOD"

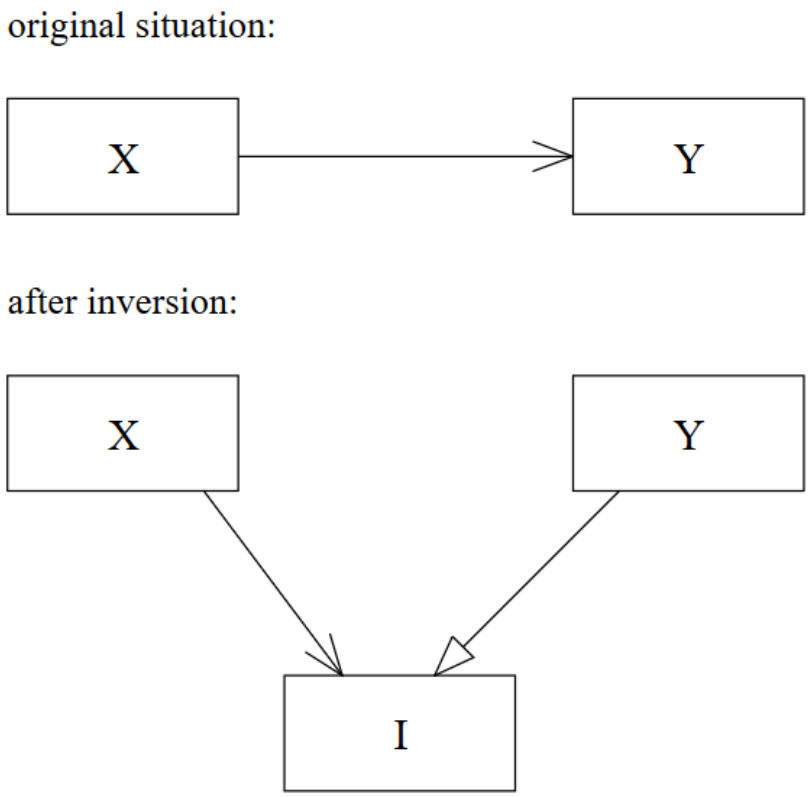
- Module implementing interface should not be forced to implement methods that won't be used
- Split interface to be more granular so that it's methods are cohesive (it always makes sense to have access to all of them)



DEPENDENCY INVERSION

„DEPEND ON ABSTRACTION NOT CONCRETO"

- high level modules should be independent of low level implementation (e.g. highlevel class responsible for printing invoices should not be aware of exact printer driver/producer)



AKADEMIA FINANSÓW I BIZNESU
VISTULA

TEST FRAMEWORKS&TOOLS

„Don't develop attachement to any one weapon or any one school of fighting"

By Miyamoto Musashi

Wstęp do programowania Java dla testerów

Remigiusz Dudek

JUNIT

- Annotations
  - `@Test` – test method
  - `@Before` – annotated method will be invoked before each test
  - `@After` – annotated method will be invoked after each test
  - `@BeforeClass` – annotated metho will be invoked once, before all tests from given class
  - `@AfterClass` – annotated method will be invoked once, after all tests from given class
- Test rules – various functionalities to be invoked before or after tests
  - `ExpectedException` – rule to handle exceptions
  - `TemporaryFolder` – rule to create temporary folder
  - `TestName` – rule to get name of the invoked tests (sometimes usefull when designing test frameworks)
- Test categories – experimental feature

JUNIT PARAMS

JUnitParams

- `JUnitParamsRunner` – special runner needed to run parameterized tests
- `@Parameters` – annotation used to either define set of parameters for given test, or to define a method that generates set of parameters for given test
- `@FileParameters` – annotation used to get parameters from file
- `@CombinedParameters` – carthesian product of parameters
- `@CustomParameters` with `@ParametersProvider` – to create custom way of providing parameters

ASSERTJ

- fluent API framework for Java assertions
- http://joel-costigliola.github.io/assertj/

TEST DOUBLES WITH MOCKITO

- `@MockitoJUnitRunner` – special runner that understands Mockito annotations (if you need to use different runner, Mockito provides factory methods to use instead annotations)
- `@Mock` – creates a mock, object with predefined behaviour
- `@Spy` – semi-mock, real methods are invoked but it is possible to stub them and verify their parameters
- Regardless of the framework you will use for mocking, do not over-do with mocks

EXTRA HOMEWORK

1. Write scoring engine for simplified Dices – given three dices (1-6) following scoring rules should be applied
   a) If all three dices indicate different number – 0 pts
   b) If two numbers are the same – 10 pts
   c) If three numbers are the same – 20 pts
2. Write a Converter class that can convert string `The value is:` … into appropriate type (Converter will have three different methods)
   1. `The value is: 102` – method should return `int 102`
   2. `The value is: true` – method should return `boolean true`
   3. `The value is: The value is:` – method should return `String The value is:`
3. `UserRegistry` – `UserRegistry` holds information about roles associated with a `User` (let's assume that `User` is uniquely identified by `name`). Create a class that gives you possibility to add/remove roles from the user. It should also be possible to check if given user has given role.